

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

2

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

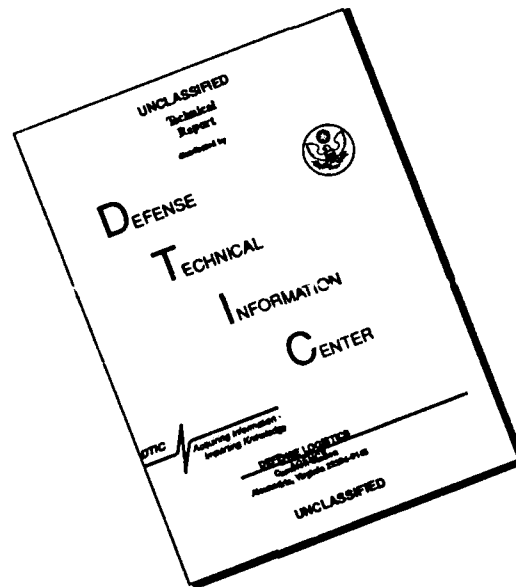
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE		3. REPORT TYPE AND DATES COVERED FINAL 01 May to 31 Oct 90	
4. TITLE AND SUBTITLE Adaptive Methods for Compressible Fluid Dynamics				5. FUNDING NUMBERS AFOSR-86-0148 61102F 2304/A3	
6. AUTHOR(S) Professor Marsha Berger					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) New York University Courant Institute of Mathematical Sciences 251 Mercer Street New York, NY 10012				8. PERFORMING ORGANIZATION REPORT NUMBER AFOSR-86-0148	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NM Bldg 410 Bolling AFB DC 20332-6448				10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFOSR-TR-87-01	
11. SUPPLEMENTARY NOTES					
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.					
12b. DISTRIBUTION CODE					
<p>Two major research efforts have been supported by this grant. The first is the development of an adaptive algorithm for hyperbolic conservation laws with simple physical geometry. This work is based on a combination of two approaches - an adaptive mesh refinement technique that concentrates computational effort where is most needed, and a high order Godunov method developed for high Mach number compressible flow. This approach has aided in the resolution of the weak von Neumann paradox in shock reflection. It was used to perform the first calculation of Kelvin Helmholtz instability along the slip line in ramp reflection off an oblique wedge. When combined with other algorithms, for example, multifluid tracking, it could categorize refraction patterns when a shock hits an oblique material interface. When combined with an elliptic grid generator, it was used to study the diffraction of a shock over an obstacle. In each of these cases, this approach to time-dependent fluid flow yielded factors of 10 to 100 improvement in efficiency over equivalent fine grid calculation with uniform resolution.</p>					
14. SUBJECT TERMS 91 3 08 047				15. NUMBER OF PAGES	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT unclassified	20. LIMITATION OF ABSTRACT SAR		

DTIC
ELECTE
MAR 14 1991
S D D

AD-A232 786

DTIC FILE COPY

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

ADAPTIVE METHODS FOR COMPRESSIBLE FLUID DYNAMICS
AFOSR-86-0148 FINAL REPORT

Marsha J. Berger

Two major research efforts have been supported by this grant. The first is the development of an adaptive algorithm for hyperbolic conservation laws with simple physical geometry. This work is based on a combination of two approaches - an adaptive mesh refinement technique that concentrates computational effort where it is most needed, and a high order Godunov method developed for high Mach number compressible flow. This approach has aided in the resolution of the weak von Neumann paradox in shock reflection. It was used to perform the first calculation of Kelvin Helmholtz instability along the slip line in ramp reflection off an oblique wedge. When combined with other algorithms, for example, multifluid tracking, it was used to compute the interaction of a supernova remnant with an interstellar cloud, and to categorize refraction patterns when a shock hits an oblique material interface. When combined with an elliptic grid generator, it was used to study the diffraction of a shock over an obstacle. In each of these cases, this approach to time-dependent fluid flow yielded factors of 10 to 100 improvement in efficiency over equivalent fine grid calculations with uniform resolution. This work was done in collaboration with Prof. Phil Colella, at Berkeley, and Dr. John Bell at Lawrence Livermore Laboratory.

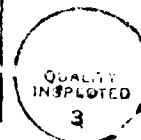
During the period of this grant, the two dimensional algorithm was developed and a paper was published. The code is currently being prepared for release to other users. The algorithm was recently extended to three dimensions. Due to the simple data structures and the use of nested grids in the same topology as the coarse grid, this involved little additional algorithm development. The most major change involved the development of a new fine grid generator. Memory usage for three dimensional calculations is at a premium. The old (two-dimensional) grid generator often resulted in overlapping subgrids or not very efficient grids (encompassing too many cells that were unnecessarily refined). After extensive experimentation, we are now using an algorithm based on edge detection, borrowed from the computer vision and pattern recognition literature, to do a smart decomposition of the underresolved regions into efficient sub-rectangles. One paper on this work will soon appear, and another is nearing completion.

The second major effort supported by this grant is the development of a Cartesian grid method to solve problems in complex geometry. This will use all of the machinery developed above, i.e. the high resolution Godunov methods and adaptive mesh refinement, along with special difference schemes that need to be developed for the irregular cells along the edge of the domain where a grid intersects a solid body. This work is in collaboration with Prof. Randall LeVeque.

Our first two approaches to the "small cell" problem at the irregular cells used the large time step wave propagation method of LeVeque, and the flux redistribution algorithm of Chern and Colella. Problems with both of these algorithms (reduced stability in the first case, and poor accuracy in the second) led

us to develop a third third approach to the "small cell" problem. We have developed a rotated difference scheme for use in the normal and tangential directions at each point in the boundary, and are currently extending it to second order accuracy. In addition to algorithm development, this projects leads naturally to research into the accuracy of difference schemes on irregular grids. This work has relevance to other types of approaches to this problem, for example, unstructured mesh algorithms for complex geometries. Currently we have a two dimensional implementation of a Cartesian mesh algorithm for time-dependent flow, including an initial implementation of the adaptive mesh refinement algorithm described above.

Accession For	
NTIS	CRASH
DTIC	and
Un	of
Justification	
By	
Distribution	
Availability	
Dist	Availability
A-1	23



Cartesian Meshes and Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations

<i>Marsha Berger</i>	<i>Randall LeVeque</i>
Courant Institute	Math Department
251 Mercer St.	University of Washington
NY NY 10012	Seattle, WA 98125

ABSTRACT

We describe a Cartesian mesh algorithm with adaptive mesh refinement for computing fluid flows in complicated geometries. Stable boundary conditions are needed at the irregular cells where the Cartesian mesh intersects the body. We develop a difference scheme that is stable even when these irregular cells are orders of magnitude smaller than the regular cells. We illustrate its performance with some computational examples solving the two dimensional Euler equations for inviscid flow.

Introduction

We describe a Cartesian mesh method to solve fluid flow problems in complicated geometries. In this approach, we keep a uniform rectangular (Cartesian) grid and allow the solid boundary to intersect the grid cells in an essentially arbitrary way. Cartesian meshes are an appealing way to simplify the grid generation problem for complex domains. Multiply connected domains and irregular geometries are only slightly more complicated than a simple domain.

Cartesian meshes have by and large been overlooked in favor of body-fitted meshes or the more recently popular unstructured meshes, but they deserve much more attention. Cartesian meshes have the advantage of allowing the use of high resolution methods for shock capturing that are difficult to develop on unstructured grids. They also allow for efficient implementation on vector computers without using gather-scatter operations (except at boundary cells). They incur little computational or memory overhead since there are no metric terms and they use far fewer pointer arrays than their unstructured counterparts. Among the few references on Cartesian mesh methods are [Clarke, Salas and Hassan; Choi and Grossman].

The major technical issue in Cartesian mesh methods is the small cell problem. Arbitrarily small cells arise at the edge of the domain where the grid intersects a body. Stable, accurate and conservative

difference schemes are needed for these cells. Moreover, the time step for a time-accurate computation should still be based on the volume of the regular cells away from the body, and not restricted by small boundary cells. Previous efforts to use Cartesian meshes have merged these small cells together until a cell with sufficient volume for stability is obtained. Clearly this loses resolution. In addition, if cell size is not taken into account in implementing finite difference schemes on irregular grids, a second order scheme can lose one or two orders of accuracy.

Our treatment of boundaries can be combined naturally with our adaptive refinement strategy using locally uniform meshes. We retain the advantages (efficiency and accuracy) of uniform grids and are able to resolve fine scale flow features induced by complex geometries. We are using the adaptive mesh refinement algorithm (AMR) described in [Berger and Colella] to achieve accuracy comparable to the body-fitted meshes, where grid points can be bunched in an a priori manner to improve the accuracy of the solution.

A more complete description of our approach to developing stable boundary conditions for irregular cells is described in [Berger and LeVeque]. Here, we only sketch the main ideas, and present computational examples for two dimensional time dependent flow in several different geometries.

One Dimensional Model Problem

We motivate the basic approach in one dimension, where we solve the equation $u_t + f(u)_x = 0$ on a uniform grid except for one small cell in the middle (see Figure 1). Let h be the cell size of the uniform grid, and αh the small cell size, $0 \leq \alpha \leq 1$. We use an explicit finite volume scheme to update all cells,

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{h} (f_{i+1/2} - f_{i-1/2})$$

at the regular cells, $i \neq 0$, and

$$u_0^{n+1} = u_0^n - \frac{\Delta t}{\alpha h} (f_{1/2} - f_{-1/2})$$

at the small cell. We want to define the fluxes $f_{\pm 1/2}$ so that the overall scheme is stable as $\alpha \rightarrow 0$.

Typical flux functions for the regular cells include Godunov's method, which for a scalar equations with $f'(u) \leq 0$ is just upwind differencing, with

$$f_{i+1/2} = F(u_i, u_{i+1}) = f(u_{i+1})$$

and the second order Lax Wendroff scheme

$$F(u_i, u_{i+1}) = \frac{(u_i + u_{i+1})}{2} + \frac{\Delta t}{2h} (f(u_{i+1}) - f(u_i)).$$

However, these definitions must be modified at the small cell, since if we define $f_{1/2} = F(u_0, u_1)$, the resulting scheme loses accuracy and becomes unstable for small α . Our approach is to define a new state

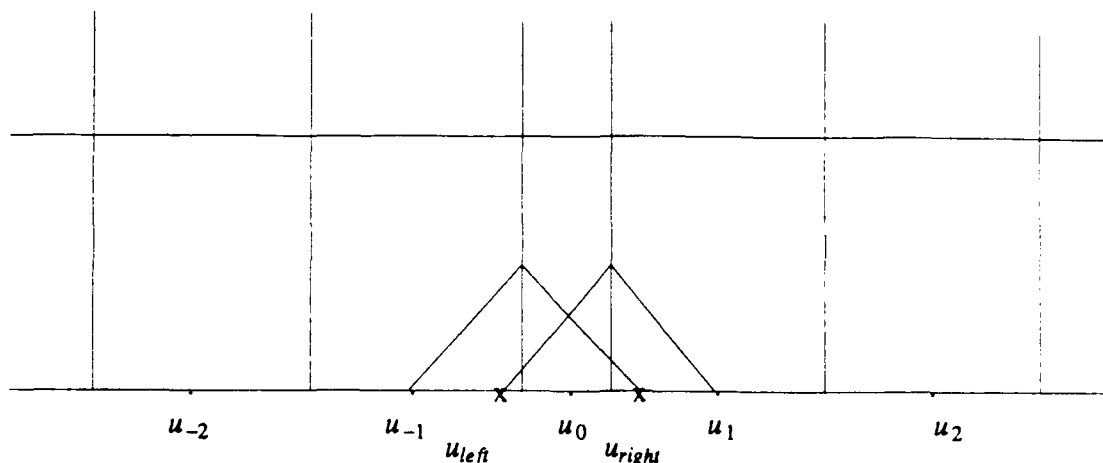


Figure 1 Special flux formulas are needed at the edges of the small cell to maintain stability and accuracy.

u_{left} and let

$$f_{1/2} = F(u_{left}, u_1).$$

Here, u_{left} is an approximation to u at distance $h/2$ from the interface. For example, u_{left} might be obtained by linear interpolation between u_{-1} and u_0 ,

$$u_{left} = \frac{2\alpha u_0 + (1-\alpha)u_{-1}}{1+\alpha}.$$

Note that $u_{left} \rightarrow u_0$ as $\alpha \rightarrow 1$, and $u_{left} \rightarrow u_{-1}$ as $\alpha \rightarrow 0$. In either of these limits the grid becomes regular again and the difference scheme reverts to the uniform scheme.

Similarly, we define a flux $f_{-1/2}$ using the left state u_{-1} and a new right state u_{right} defined by interpolating u to a point a distance $h/2$ to the right of this interface, e.g.

$$u_{right} = \frac{2\alpha u_0 + (1-\alpha)u_1}{1+\alpha}.$$

We then set

$$f_{-1/2} = F(u_{-1}, u_{right}).$$

It can be shown for the equation $u_t = u_x$ that if u_{left} and u_{right} are obtained by linear interpolation then the resulting scheme is stable as $\alpha \rightarrow 0$, using a time step Δt that satisfies the CFL condition for the regular grid, $\frac{\Delta t}{h} \leq 1$. This results holds for both upwind differencing and Lax Wendroff. However, for Lax Wendroff, a more accurate procedure would be to use quadratic interpolation for u_{left} and u_{right} . In this case, a combination of theoretical and numerical results show that if the additional point used in the interpolation is the upwind point then the scheme is stable as $\alpha \rightarrow 0$, but if the downwind point u_{-1} is used, then the CFL limit is reduced to $1/2$.

Stable Difference Equations for Two Dimensional Irregular Cells

For two dimensional calculations, we again need a new difference scheme to compute fluxes at the edges of the irregular cells adjacent to the boundary. The boundary of a solid body is represented by a piecewise linear segment in each cell, so that the irregular cells can have either 3, 4 or 5 sides. Our approach uses locally normal and tangential coordinate directions to define left and right states for a Riemann problem at each cell edge. This fits naturally with the MUSCL scheme used in the interior of the domain in our calculations. However, we generalize the pointwise approach in the one dimensional case, and use a conservative average of the solution over a box a distance h in the appropriate direction away from a cell edge. For example, in Figure 2 the state q_l is obtained using an area-weighted average of the values $u_{i,j-1}$ and $u_{i,j}$ that intersect the box from the regular grid. In an analogous way we obtain the state q_r . These values are then rotated into a frame of reference that is tangent to the boundary, and a one dimensional Riemann problem in the tangential direction is solved. This gives the flux f_ξ . This procedure is repeated for the dashed boxes in Figure 2 that are normal to the boundary, giving a flux f_η in the η direction. (The part of the box that lies outside the domain is interpolated from \bar{q}_k , see Figure 3). The final value of the flux at the vertical interface is a linear combination $f_\xi \cos\theta + f_\eta \sin\theta$, where θ is the angle the boundary makes with the grid. For a boundary with curvature, we determine these directions using the boundary segment of the cell with the smaller area adjacent to the interface. This helps retain stability for the

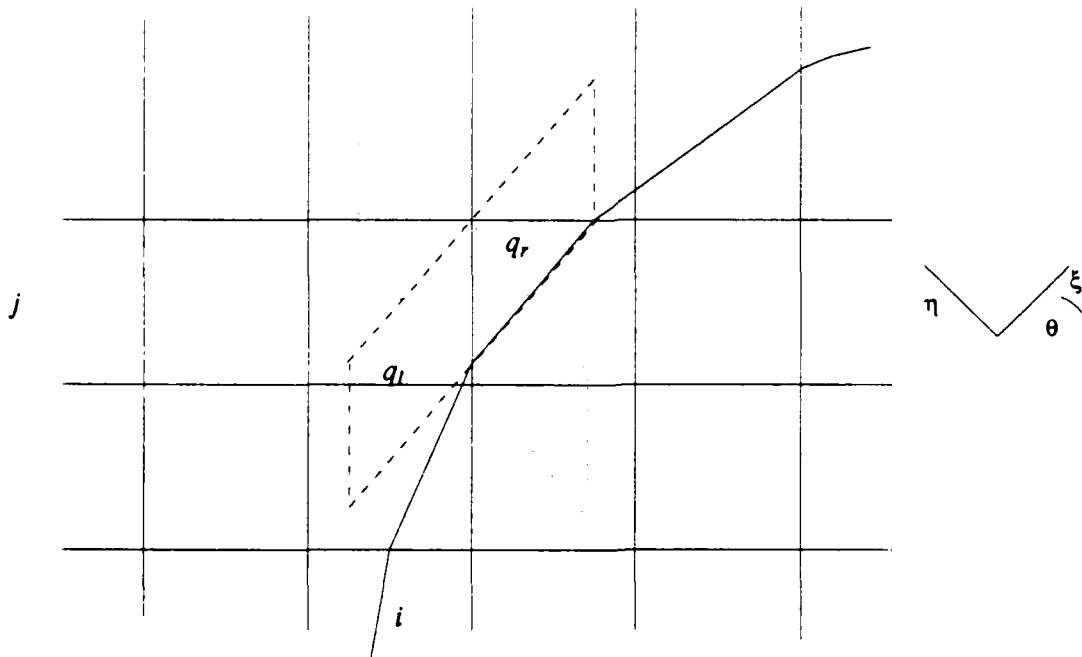


Figure 2 shows a schematic of the rotated difference scheme used to define the vertical flux.

smaller cells, by maintaining a certain cancellation property of our flux definitions, described more completely in [Berger and LeVeque]. Related work using rotated difference schemes has been done by [Jameson; Davis; Levy, Powell and van Leer].

At the solid wall boundary itself, the flux can be determined more simply, using only boxes normal to the boundary as shown in Figure 3. First we obtain a value q_k for the box interior to the domain, using area weighted averages, and rotate the velocities into the boundary coordinate frame. A boundary Riemann problem is solved between q_k and \bar{q}_k (with negated normal velocity), to satisfy the the boundary conditions of no normal flow.

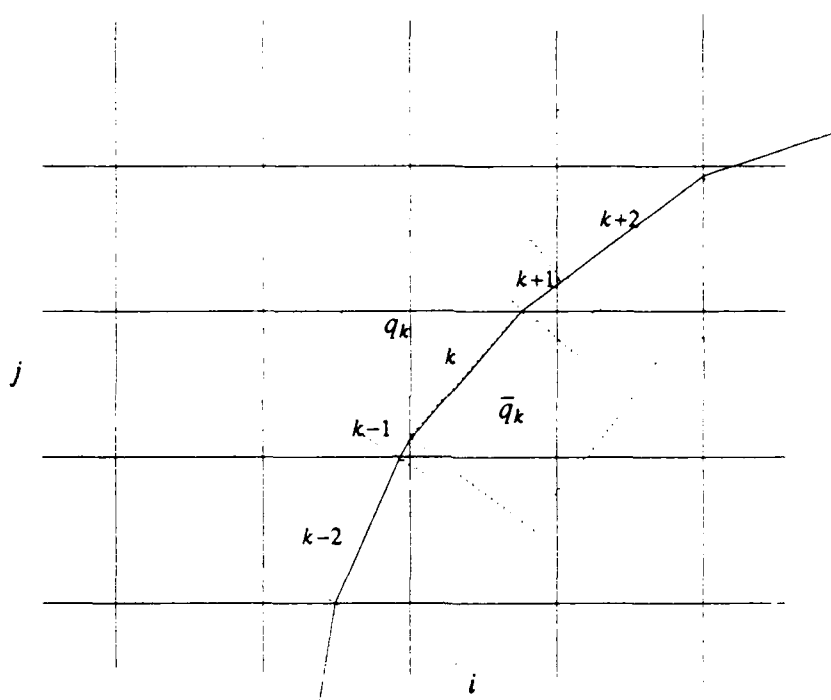


Figure 3 indicates the scheme used to determine the boundary flux.

In this case, if the solid wall boundary happens to align with the Cartesian grid, the scheme reverts to the usual first order Godunov method. To improve the scheme to second order, following the MUSCL approach as described in [Colella], we need to introduce limited slopes in the solution reconstruction phase, and tangential derivatives for predicting states at the cell edges. These steps are also necessary to improve the stability limit for Godunov's method from 1/2 to 1. Work on these improvements is continuing. Referring to Figure 3, we add a tangential derivative f_ξ to the state q_k for the normal Riemann problem, with $f_\xi = f(u(k, k+1)) - f(u(k-1, k)) / l_k$, where l_k is the length of the k^{th} boundary segment. The state $u(k, k+1)$ comes from solving a Riemann problem in the tangential direction at the interface between cells k and $k+1$. As before, the stencil for this Riemann problem must be enlarged beyond the adjacent cells to maintain stability. For example, the right state at this interface is not just the value q_{k+1} , but a linear

combination of the solution in several boxes, q_{k+1} and q_{k+2} , up to a distance h away from the interface. The left state at this right interface can be taken to be the value q_k since the length of that cell's boundary segment is larger than h . This same procedure is used to include tangential derivatives in the normal box Riemann problems for interior cell edges. This procedure alone improves the CFL limit to 1. It remains to incorporate monotonized slopes into the scheme in order to achieve second order accuracy.

While the overall scheme at the boundary involves twice as many Riemann problems as the ordinary MUSCL scheme, it is fully vectorizable. The coefficients in the interpolations for the left and right states are fixed for the duration of the integration, and are not dependent of the properties of the solution at each step. In numerical experiments in two dimensions, this scheme remains stable for cell areas that are orders of magnitude smaller than the regular cell areas (down to the round-off level). In essence, our method can be viewed as a technique for defining fluxes on an irregular grid by a very local mapping to a regular grid. This viewpoint may prove useful in defining higher order methods on unstructured grids.

Computational Example

We illustrate this by computing time dependent flow around a cylinder. The initial conditions are an incident shock traveling at Mach 2.81. We use a simple MUSCL scheme to advance the flow field in the interior of the domain. Figure 4 shows a contour plot of the flow field, as well as a plot of density as a function of arclength around the cylinder. The only cells drawn on the contour plot are the irregular cells from the Cartesian grid that intersect the body. Note the smoothness of the arclength plot despite the irregularity of the grid around the body. This example was computed using the local mesh refinement of [Berger and Colella]; the location of the rectangular fine grids is indicated on the contour plot as well.

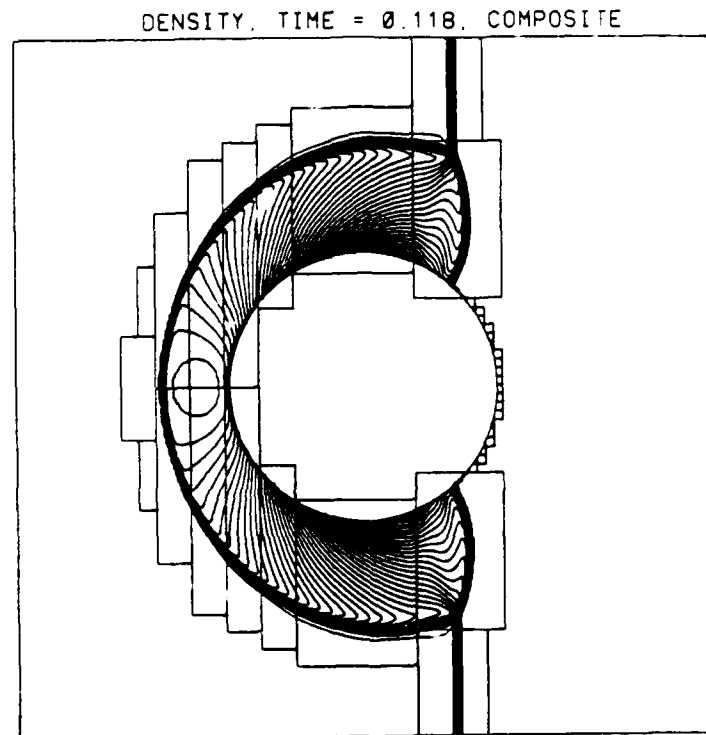
References

- M. Berger and P. Colella, "Local Adaptive Mesh Refinement for Shock Hydrodynamics". J. Comp. Phys. 82, 1989.
- M. Berger and R. LeVeque, "Stable Boundary Conditions for Cartesian Grid Calculations". Proc. Symposium on Computational Technology for Flight Vehicles, Pergamon Press, Nov. 1990, to appear. ICASE Report No. 90-37, May, 1990.
- S. Choi and B. Grossman, "A Flux-Vector Split, Finite Volume Method for Euler's Equations on Non-Mapped Grids", AIAA Paper 88-0227, Reno, Nevada.
- D. Clarke, M. Salas and H. Hassan, "Euler Calculations for Multielement Airfoils Using Cartesian Grids", AIAA Journal 24(3), 1986.
- P. Colella, "Multidimensional Upwind Methods for Hyperbolic Conservation Laws", J. Comp. Phys. 87, 1990.

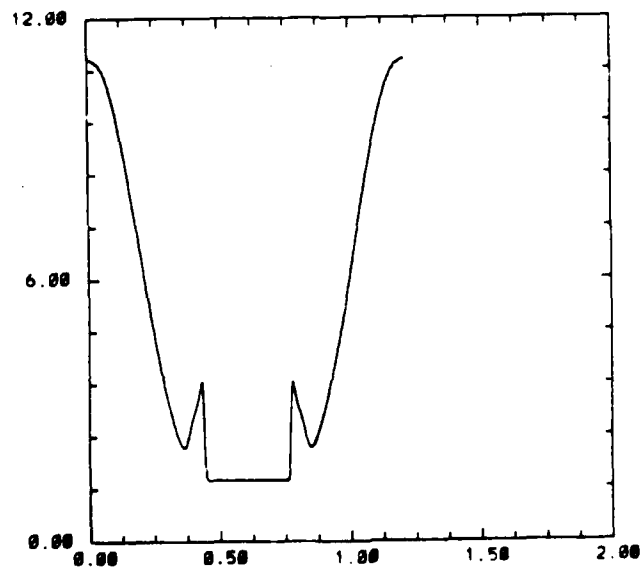
S. Davis, "A Rotationally Biased Upwind Difference Scheme for the Euler Equations", J. Comp. Phys. 56, 1984.

D. Levy, K. Powell, and B. van Leer, "An Implementation of a Grid Independent Upwind Scheme for the Euler Equations", AIAA Paper 89-1931-CP, Buffalo, NY.

A. Jameson, "Iterative Solution of Transonic Flows over Airfoils and Wings, Including Flows at Mach 1", Comm. Pure Appl. Math 27, 1974.



(a)



(b)

Figure 4 (a) Density contours of the flow around the cylinder. (b) Density around the boundary of the cylinder.

*To appear, IEEE
Trans. Sys., Man, Cyber.*

**An Algorithm for Point Clustering
and Grid Generation**

by

**Marsha Berger
Isidore Rigoutsos**

**Technical Report No. 501
Robotics Report No. 229
April, 1990**

**New York University
Dept. of Computer Science
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, New York 10012**

This work was supported in part by the Air Force Office of Scientific Research under Contract No. AFOSR-86-0148, by the Department of Energy Contract No. DE-FG02-88ER25053, and by the NSF Presidential Young Investigator Award ASC-8858101. The authors would like to thank Cray Research and Grumman Aerospace for matching funds in support of the PYI award.

An Algorithm for Point Clustering and Grid Generation

Marsha Berger

Isidore Rigoutsos

Courant Institute of Mathematical Sciences
251 Mercer St.
New York, NY 10012

ABSTRACT

We describe a new point clustering algorithm, and its application to automatic grid generation, a technique used to solve partial differential equations. Algorithms from the computer vision and pattern recognition literature are used to partition points into a set of enclosing rectangles. We show examples from two dimensional calculations, but the algorithm generalizes readily to three dimensions.

1. Introduction

This paper presents a new point clustering algorithm and its application to automatic grid generation. The algorithm clusters the points into distinct rectangles such that neighboring points are in the same rectangle (as much as possible), and all points are contained in some rectangle. The application is best illustrated by an example. We are solving a partial differential equation using finite difference techniques. The difference equations are first solved on the uniform coarse grid in Figure 1. An "error estimation" procedure [Olinger] is then used to flag grid points that need to be in a finer grid, which is just a smaller rectangular grid with finer mesh spacing. Figure 1 shows the flagged coarse grid points as well as the new fine grids that together contain all the flagged points. This procedure is usually employed recursively, leading to a nested sequence of increasingly fine, locally uniform subgrids which are superimposed on the underlying base grid.

The grid generation problem then is to define a set of rectangles that enclose all the flagged points. Certain factors make a huge difference in the performance of these grids in solving the pdes.

[1] *There should be as little unnecessarily refined area as possible.*

Since the cost of the numerical integration procedure is proportional to the area of the rectangle, the smaller the better. Figure 1 gives an example of a set of flagged points for which a few patches lead to much less refined area than refining the whole grid. This gain in efficiency is the purpose of adaptive

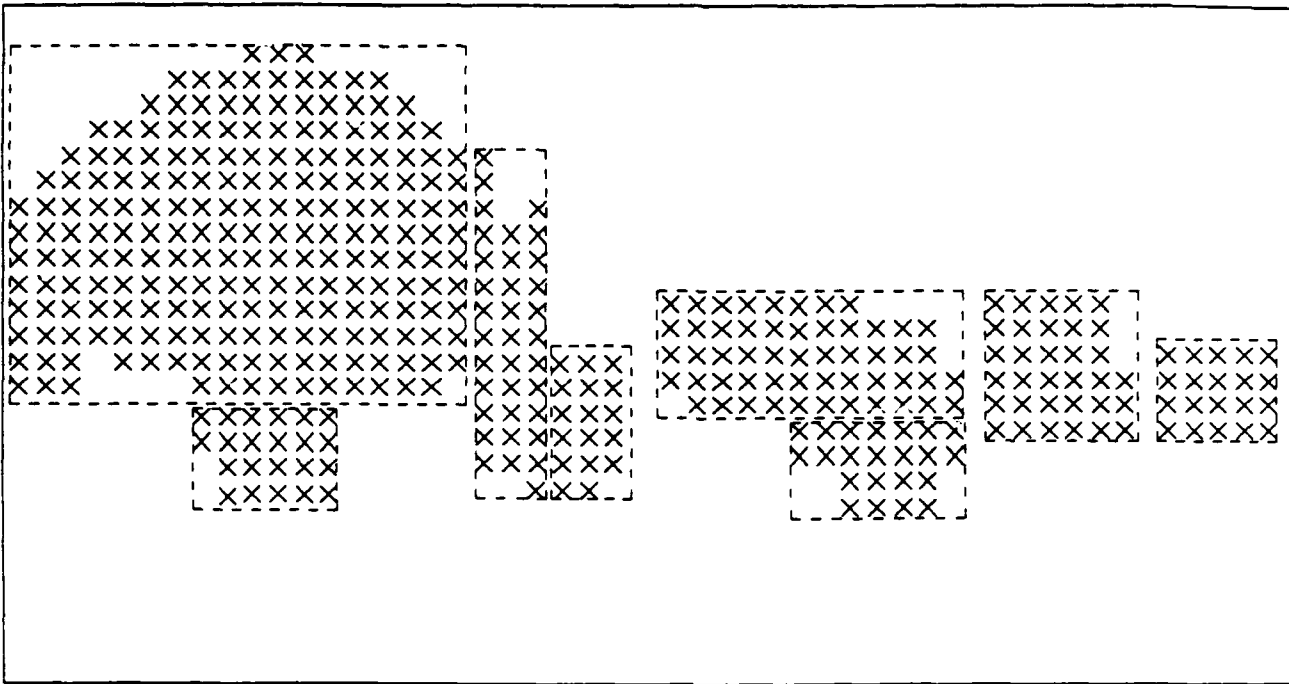


Figure 1 shows a coarse grid with rectangular subgrids around the flagged points. The coarse grid points with high error are marked with an "x".

methods. Some unnecessarily refined area (or inclusion of non-flagged coarse grid points in a new rectangle) is inevitable, since we are restricted to using rectangles. In addition, for numerical reasons the rectangles are oriented with the base grid rectangle. This is true even if the flagged points lie on a diagonal of the coarse grid, and could be perfectly enclosed by a rotated rectangle. (However, an algorithm that uses rotated rectangles is considered in [Berger]). Along these lines, if several rectangles are used to enclose the flagged points, their overlap should be minimal.

Another criterion for generating these rectangles is:

- [2] *There should be as few rectangles as possible.*

At the other extreme, we could put one tiny rectangle around each flagged point. Many of these tiny rectangles would share a common boundary segment. However, there is boundary overhead associated with each rectangular subgrid that should also be minimized, along with the area. In addition, these procedures will be used on vector processors, which favors larger vector lengths and therefore larger rectangles. (We could worry further about this, for example by trying to maximize the length in a particular coordinate direction, but we will not consider such machine specific details here).

The third criterion is the most difficult to pinpoint.

[3] *The rectangles should "fit" the data.*

This is hard to make absolutely precise, but for example, if a person were to put the rectangles around the points by hand, using whatever clustering or partition the brain uses, it would "look right". Although this is not essential for accurate numerical integration on the rectangles, we prefer the adaptively generated subgrids to be pleasing. Finally,

[4] *The algorithm should be fast.*

This algorithm is repeated every few timesteps, or hundreds of times during any particular numerical simulation, and should therefore be fast relative to the time needed to take a time step on the resulting grids.

Our solution to this rectangle-fitting problem uses algorithms from the pattern recognition and computer vision literature. A combination of signature arrays and zero crossings of second derivatives is used to partition the flagged points into rectangles. Our examples are all in two dimensions, but the algorithm generalizes readily and has proven effective in three dimensions too. Before describing our algorithm, we give a little background and discuss some other approaches we tried and discarded.

1.1. Previous Algorithms

Our previous algorithms for this problem can be summarized as being of two main types: bottom-up or top-down. The top-down approach is based on a bisection method. It can be viewed as a form of divisive hierarchical clustering [Duda and Hart]. Initially, the flagged points of the grid are surrounded by a single rectangle, and its *efficiency* is computed. Here, we define the efficiency of a grid as the ratio of flagged points to the total number of coarse grid points in the new rectangle. This is one of the key parameters behind our algorithm, and it is easily computed. If the efficiency is above a preselected threshold, the rectangle is accepted and the algorithm stops. Otherwise, we bisect the longest direction of the rectangle, and generate two new subgrids. This process is repeated recursively on each of the two subgrids. When the algorithm terminates, all of the subgrids are guaranteed to have acceptable efficiency ratings. However, hierarchical clustering methods are known to create clusters even if no natural clusters exist [Anderberg; Hartigan; Jain and Dubes]. In addition, since the bisection uses no information about the locations of the flagged points, a non-optimal grid hierarchy is generally created. To alleviate this, we usually follow the bisection step with a merging step, where neighboring subgrids are merged into larger subgrids if the result continues to be acceptably efficient. This merging step is what leads to the problem of overlapping grids.

The bottom-up approach starts at the grid point level. The flagged points are organized into a minimal spanning tree, so that each point is connected to its nearest neighbor. Neighboring branches of the tree are merged, either one point at a time, or a front at a time, as long as the resulting grid is efficient. Although philosophically appealing, this algorithm actually performs much worse than the top-down bisection algorithm. A fundamental problem is the non-uniqueness of the minimal spanning tree. Also, the algorithm suffered from the hill-climbing problem of getting stuck in local minima; it was very sensitive in the beginning steps of the algorithm to the initial direction of growth of the clusters, and tended to stop prematurely, although larger and acceptably efficient grids were just several branches away. In that case, it had to be followed by a merging procedure as well.

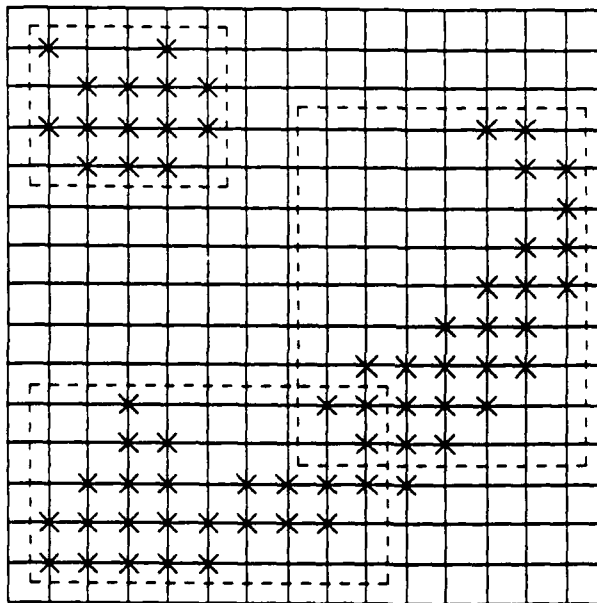


Figure 2 Nearest neighbor clustering is sufficient in simple cases, e.g., the top left cluster.

In practice, both approaches were preceded by a "nearest neighbor" clustering algorithm. The purpose of this was to separate flagged points when possible into isolated islands (see Figure 2). This sometimes produces acceptable clusters by itself, but fails to help when the flagged points formed elongated, curved shapes. Thus, it was followed by either the bisection or minimal spanning tree algorithm. Summarizing, both of these algorithms produced less than optimally efficient grids that overlapped too much. Better grids were easily created by hand.

2. Towards an Efficient Algorithm

In a more general form, the grid generation algorithm should cluster a set of m flagged points into k clusters, where k is either specified a priori or is determined by the algorithm itself. This special type of clustering is called partitioning [Anderberg]. Our first approach considered the question of how to choose a

set of k "seed points" around which the k clusters would be built.

2.1 A First Approach: Local Maxima in Two-Dimensional Grids

Initially, each of the grid points is given a value: "1" for the flagged, and "0" for the non-flagged ones. These grid values, viewed as a binary image, $I(x,y)$, are then preprocessed by convolving it with either an "averaging" or a "low-pass" filtering template, (see Figure 3) [Ballard and Brown; Horn; Levine]. This operation results in a non-convex function, $\hat{I}(x,y)$, whose local maxima, determined by the Sobel operators [Ballard and Brown; Levine] of Figure 4, compose the set of "seed points" around which we build the clusters.

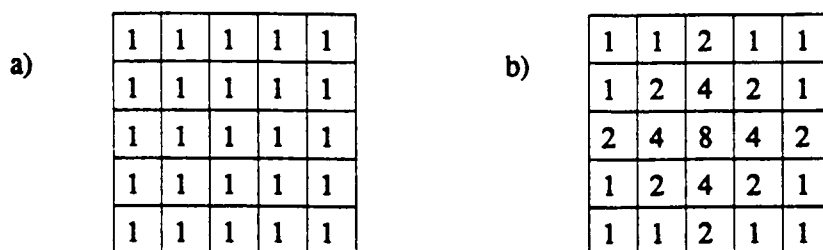


Figure 3 shows the filtering templates: (a) Averaging (b) Low-Pass.

Three partitioning algorithms were tested using the seeds found above: the standard k-means algorithm, its converging variant, and a k-means variant where no updating of the centroids takes place [Anderberg; Hartigan]. These algorithms are outlined in the Appendix.

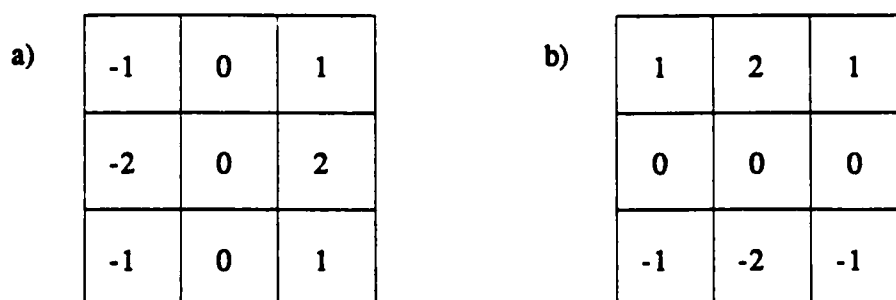


Figure 4 shows the Sobel gradient operators: (a) $\partial/\partial x$ (b) $\partial/\partial y$.

Figures 5 and 6 show graphically the output of the three algorithms on two sample data sets. The three algorithms exhibit the same overall behavior. The seeds are sensibly chosen, but in all the test cases the resulting grids overlap excessively. We attribute this problem to the large number of seeds discovered by this method. The next method tries to reduce the number of seeds, keeping only the best, and thus hopefully reducing the overlapping.

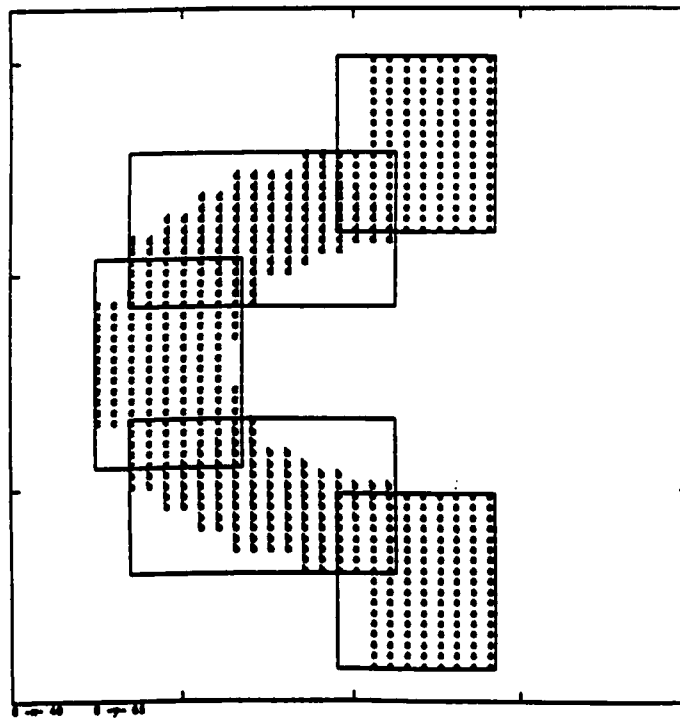
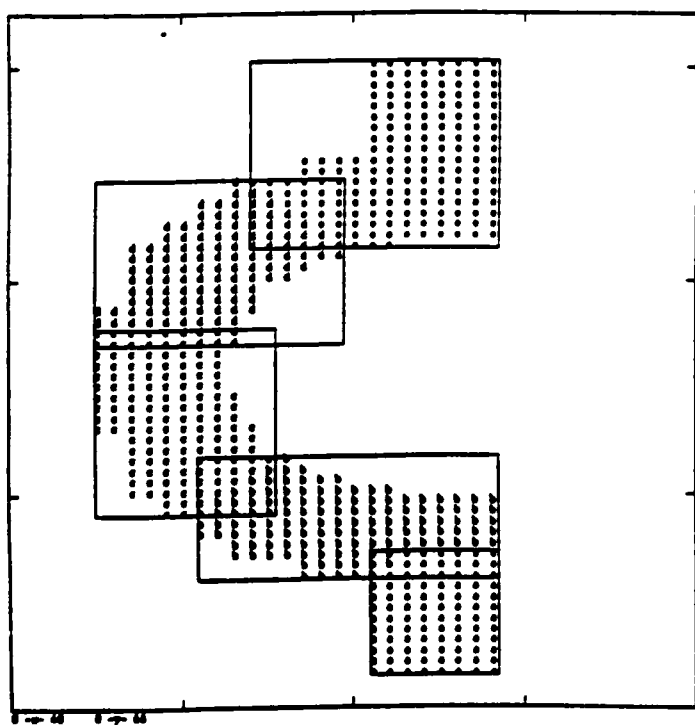
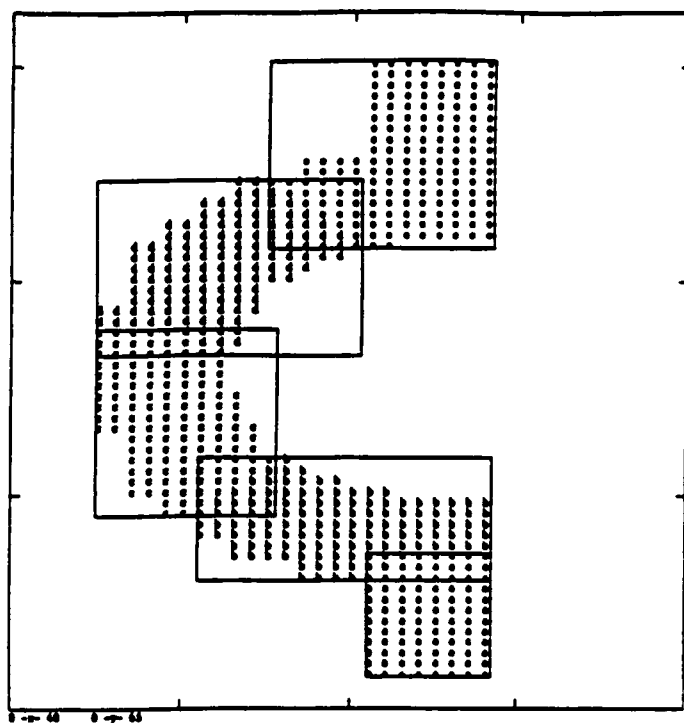
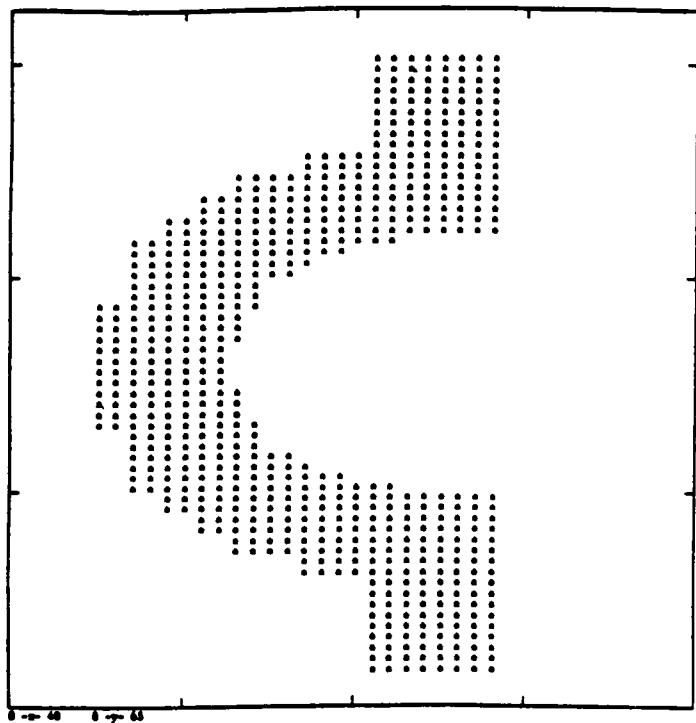


Figure 5 The seeds are local maxima of a two dimensional function.

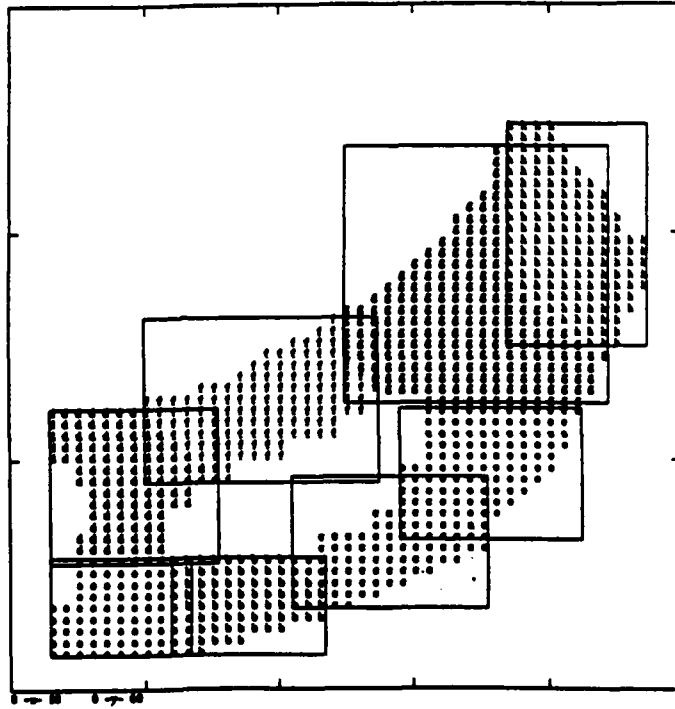
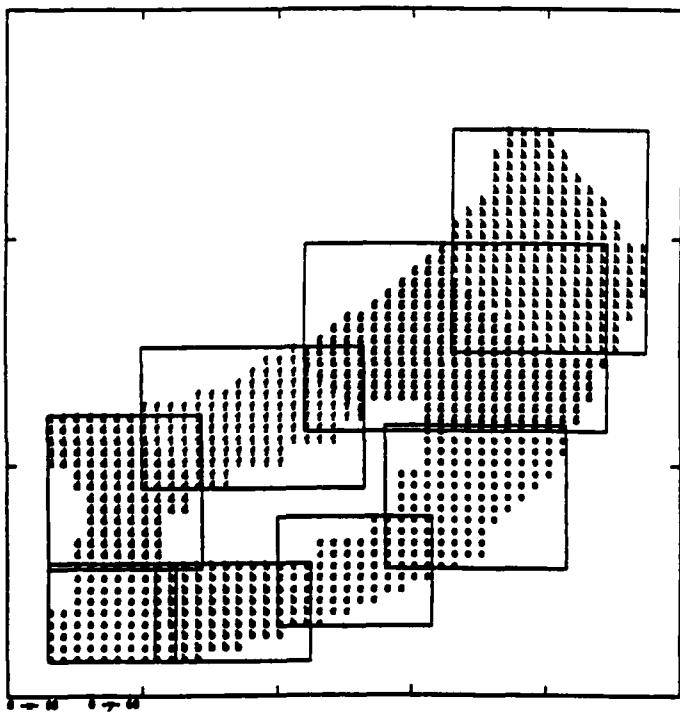
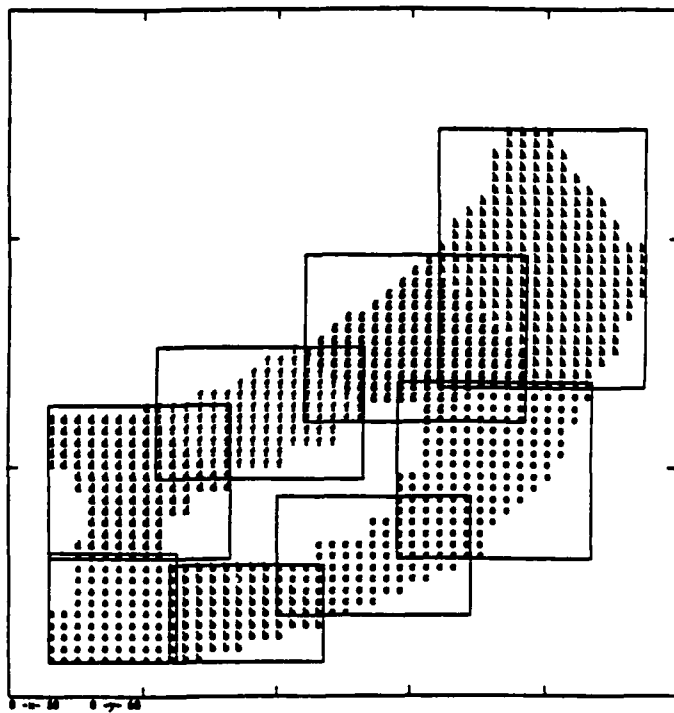
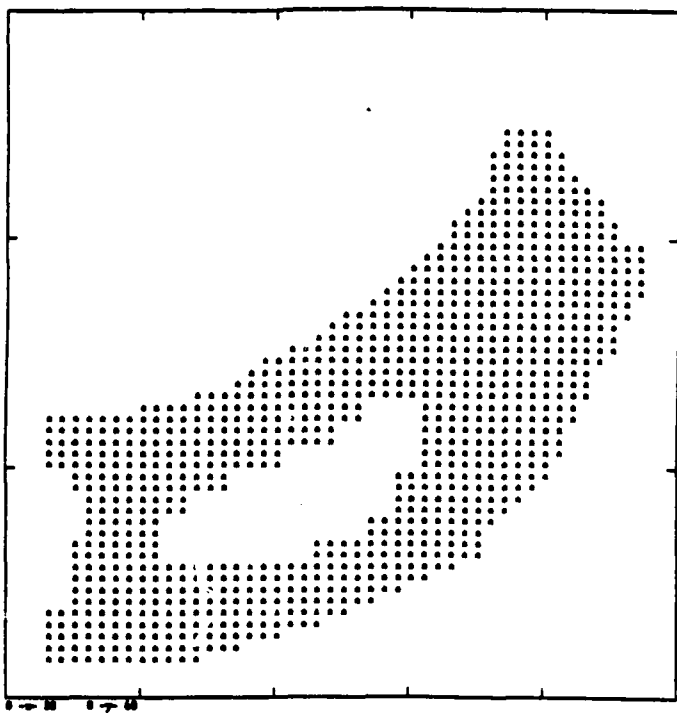


Figure 6 The seeds are local maxima of a two dimensional function.

2.2 A Second Approach: Local Maxima in Signature Arrays

Signatures have been known in computer vision and pattern recognition for many years [Ballard and Brown; Horn; Levine]. Able to encapsulate gross characteristics of a shape, and computationally simple, signatures proved very useful for establishing preliminary landmarks in images; these landmarks in turn led to a subsequent reduction of the search effort. Given a continuous function, the horizontal and vertical signatures, $H(x)$ and $V(y)$ are defined as

$$H(x) = \int_y f(x, y) dy$$

and

$$V(y) = \int_x f(x, y) dx$$

respectively.

First, the horizontal and vertical signatures of the image are computed. The resulting one-dimensional arrays are "smoothed" [Horn] using the template in Figure 7, and subsequently searched for local maxima. Let M and N be the two sets of maxima. After discarding those tuples of the Cartesian product $M \times N$ that correspond to non-flagged regions, we are left with precisely the coordinates of the starting seeds.

1	2	1
---	---	---

Figure 7 shows the one-dimensional smoothing template.

With this choice of seeds, we again employed the three partitioning techniques (k-means, converging k-means, and the no updating variant). Figures 8 and 9 show the results. This algorithm resulted in fewer subgrids and reduced overlapping, making this approach superior to using the local maxima of $\hat{I}(x, y)$ for the seed points. Unfortunately, the generated subgrids were still not the most efficient ones; better choices were clearly possible. Some observations based on extensive experimentation could still be made: 1) the non-converging variants outperformed the converging k-means, 2) overlapping was minimal when no updating of the centroids occurred, and 3) the distance metric (Manhattan Block vs. Euclidean) had no appreciable effect on the results.

Apparently, the use of local maxima in signatures did not capture enough of the underlying structure. In the next section, we use signatures in a different way to partition the flagged points into clusters.

3. The Algorithm

Our best algorithm uses ideas related to edge detection. One of the many approaches to the edge detection problem is the one suggested by [Marr and Hildreth]. Based on the psychophysical and neurophysiological experiments of [Campbell and Robson], the method consists of first convolving the original

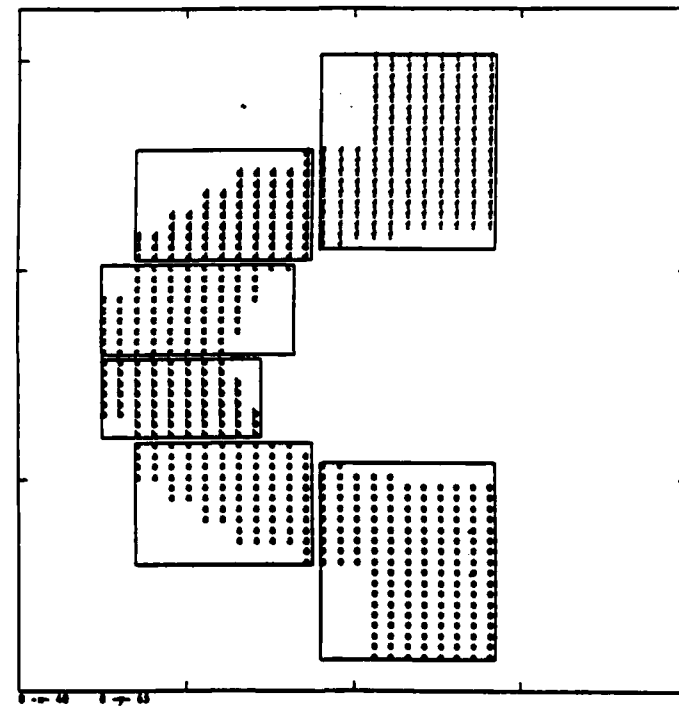
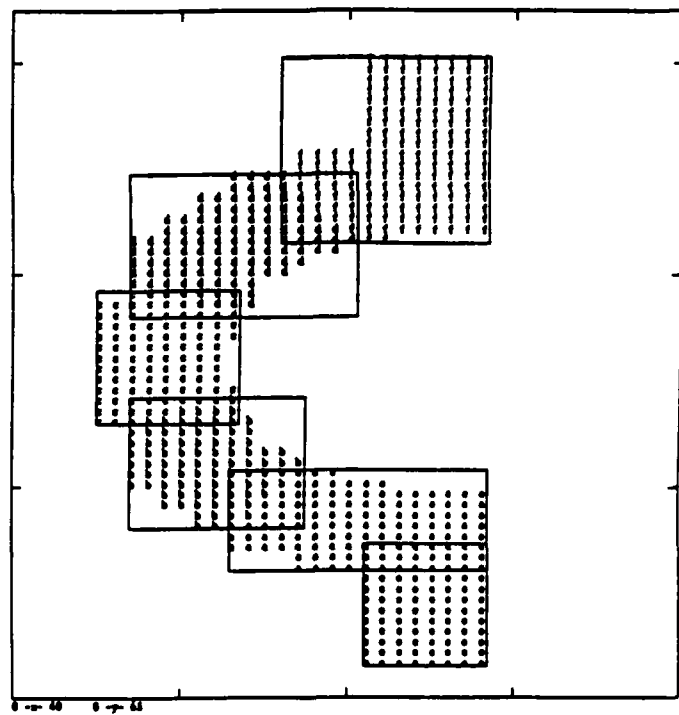
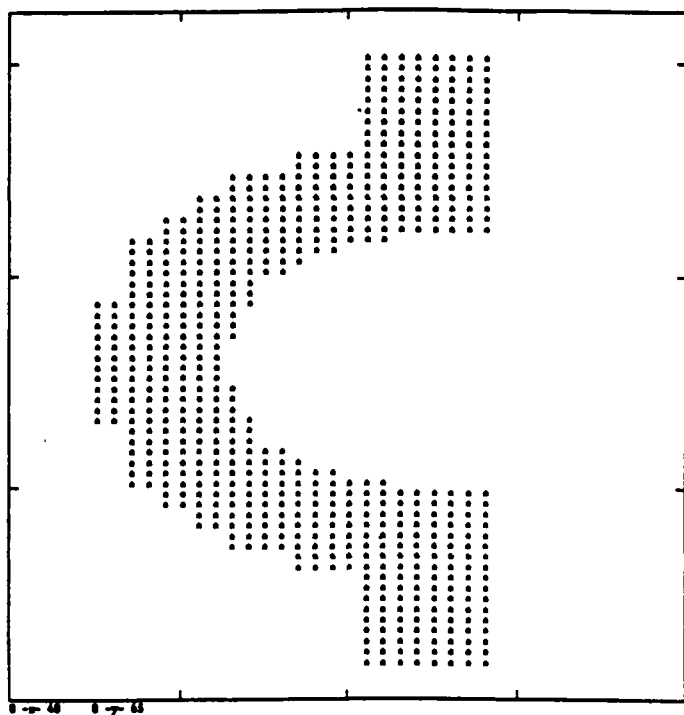


Figure 8 The seeds are chosen from local maxima of signature arrays.

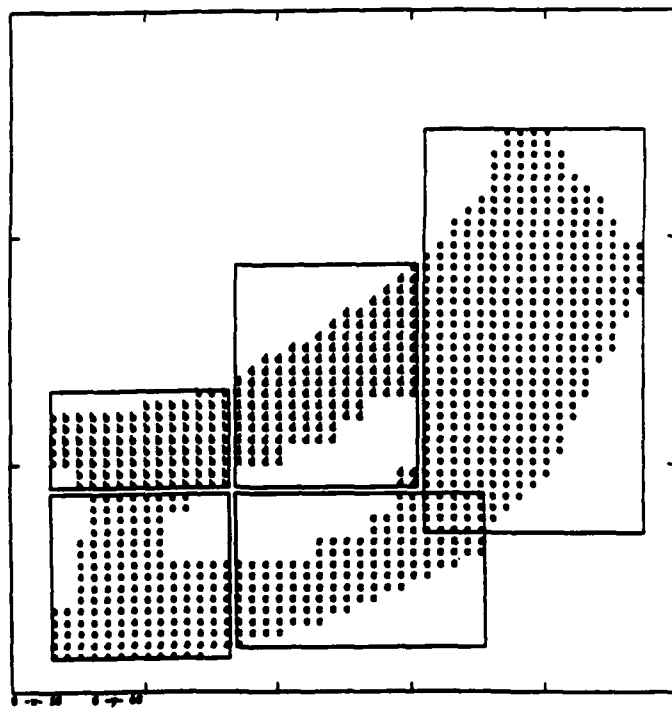
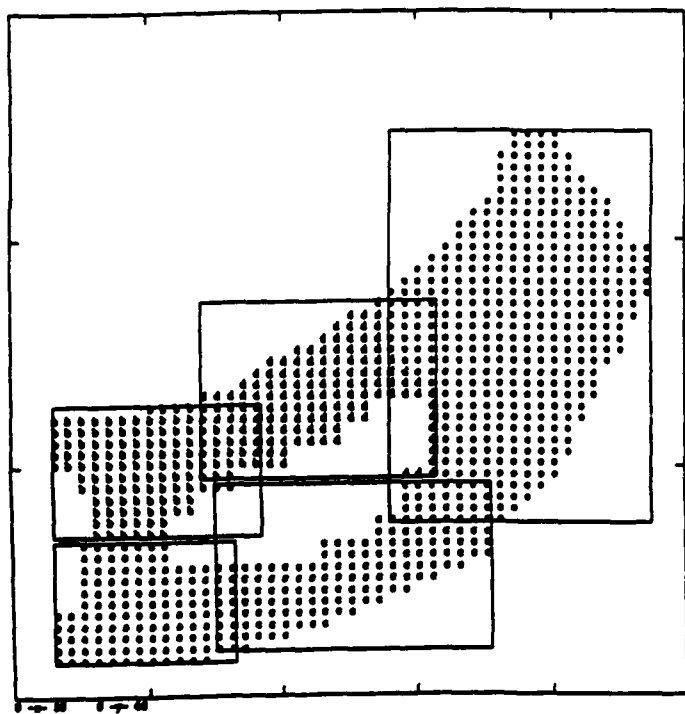
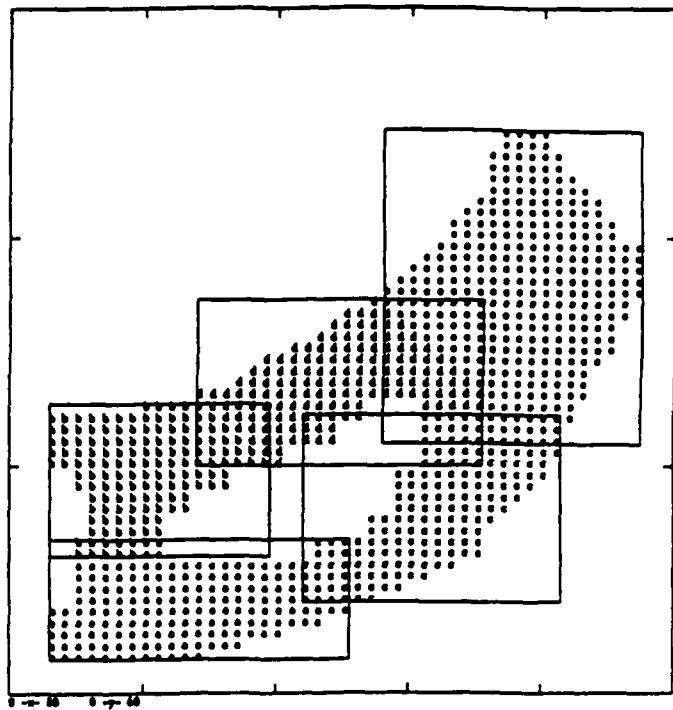
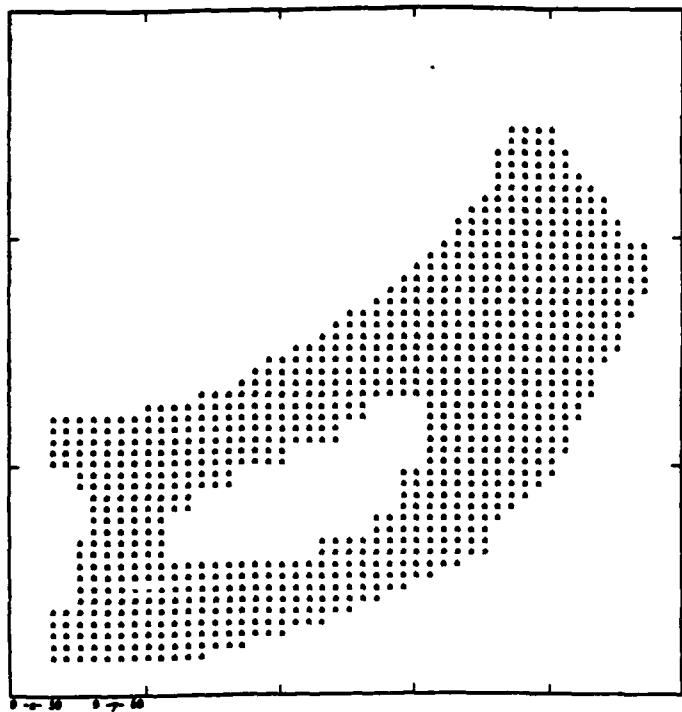


Figure 9 The seeds are chosen from local maxima of signature arrays.

image against a Gaussian kernel and then computing the Laplacian of the result; the intensity discontinuities are associated with those positions where the Laplacian is equal to 0 (zero crossings).

In what follows, the input grids are viewed as binary images in the sense of paragraph 2.1. The edges will now be located at those positions of the grid where a transition from a flagged point region to a non-flagged one occurs. The most prominent such transition represents a "natural" line with respect to which the original grid can be partitioned. For the example depicted in Figure 10, the line (e) represents such a transition.

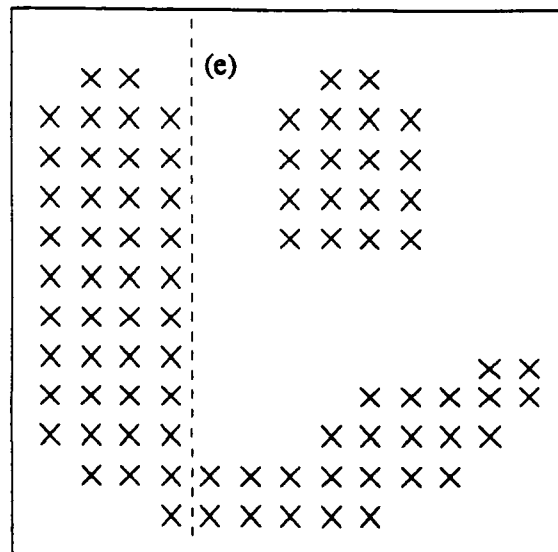


Figure 10 The rectangle is partitioned at line (e); after this, an isolated cluster can be detected for further partitioning.

The problem is that of determining such a transition. The Marr-Hildreth method, although a potential candidate, cannot easily be employed towards this end for two reasons. First, its output is a collection of Boolean values at the different grid locations: TRUE if a zero crossing exists at that location, FALSE otherwise; these Boolean values need to be combined in order to generate an actual edge, a not so trivial task. Second, since the Laplacian operator is isotropic, the "natural" line with respect to which the original grid could be split will not, in general, be parallel to the sides of the grid.

Signatures again hold the answer: the idea is to look for zero crossings in the second derivative of a signature. This idea borrows from both the signature approach and the Marr-Hildreth method, except that we do not convolve with a Gaussian filter.

In general, there is more than one zero crossing in a given signature array. For our purposes however, we only select one zero crossing at a time; it corresponds to most prominent edge, and its location is determined by searching both the horizontal and vertical signature arrays for the zero crossing corresponding to the largest local change of values. This is indicated in Figure 11, where the row (resp. column)

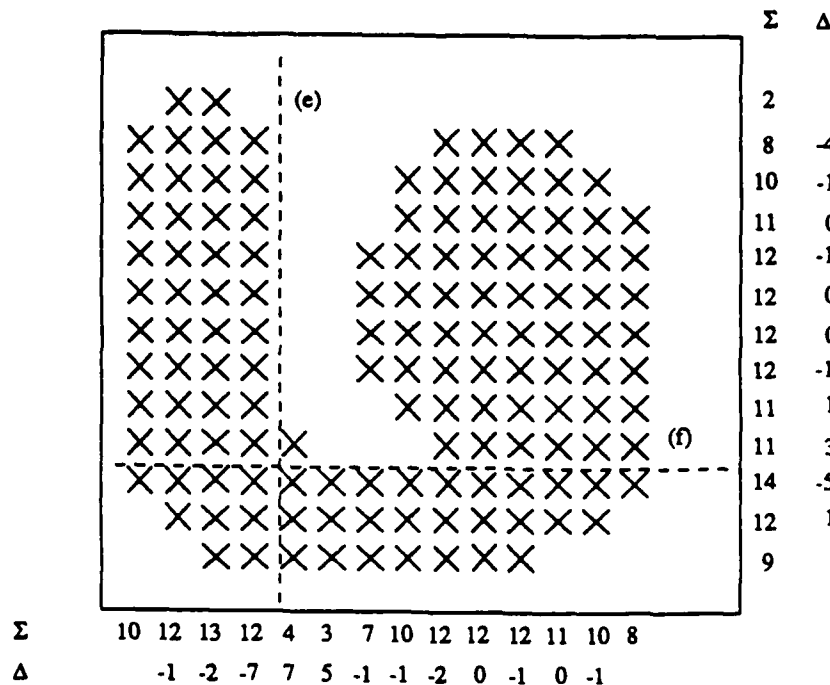


Figure 11 The steepest zero crossing occurs at line (e), and makes the most efficient rectangles.

labelled Σ contains the horizontal (vertical) signature, and the row (column) labelled Δ contains the Laplacian of the signature in the appropriate direction.

The input grids are also expected to contain isolated regions of flagged points (islands), making it necessary that both signature arrays be first searched for chains of 0's, or "holes". This can be seen in Figure 10. The occurrence of such holes provides obvious choices for splitting the input grid into a number of subgrids, and is exploited before any attempt at locating zero crossings is made. If any holes are found, we do not attempt to compute any zero crossings. We proceed by applying the above steps to only those of the generated subgrids that are still inefficient.

We now give a high level description of the actual algorithm, followed by sample runs in Figures 12 to 16 on a number of real and synthetic problem cases.


```
BEGIN
  i = 1;
  while ( i <= number_of_rectangles ) do
    if (rectangle efficiency < threshold) then
      compute signatures ;
      find the best place to split  $R_i$  ( either a hole or edge ) ;
      if ( found a place to split ) then
        split rectangle in two ;
        append new rectangle to end of list of rectangles ;
      else
        i = i + 1 ;
      endif
    else
      i = i + 1 ; ( consider the next rectangle on the list )
    end if
  end while
END
```

4. Comments on the Algorithm and its Performance

While most of the time the algorithm performs exceptionally well, sometimes it generates anomalous and/or non-optimal rectangles. As we will see, many of these can be eliminated by simple changes in the algorithm. The anomalies fall into several general categories, which we illustrate by picture. The basic algorithm description has two exit points: a rectangle is "accepted" either because its efficiency is already above threshold, or because it cannot be further split using either of the two methods (i.e. holes and inflection points). As a result, the efficiency of some of the generated subgrids may be below the preset threshold. This is the case with all grids associated with regions that form an angle with the horizontal. The inefficiency approaches its maximum as angles approach 45 degrees.

Another problematic grid is the one appearing in Figure 17: as can be seen neither the horizontal nor the vertical signatures will contain any holes or zero crossings, and this will be true for all non-zero values of a, b . In all such cases the bounding rectangles will have an efficiency of precisely 50%. An ordinary bisection step could be easily incorporated here to increase the efficiency to 100%. A similar arrangement of flagged points which we have encountered in our experiments (see Figure 18) leads to another kind of non-optimal choice. One way around this is to use weighted second derivatives, scaling the Laplacian by the number of flagged points. This leads to a correct choice for the zero crossing in Figure 18.

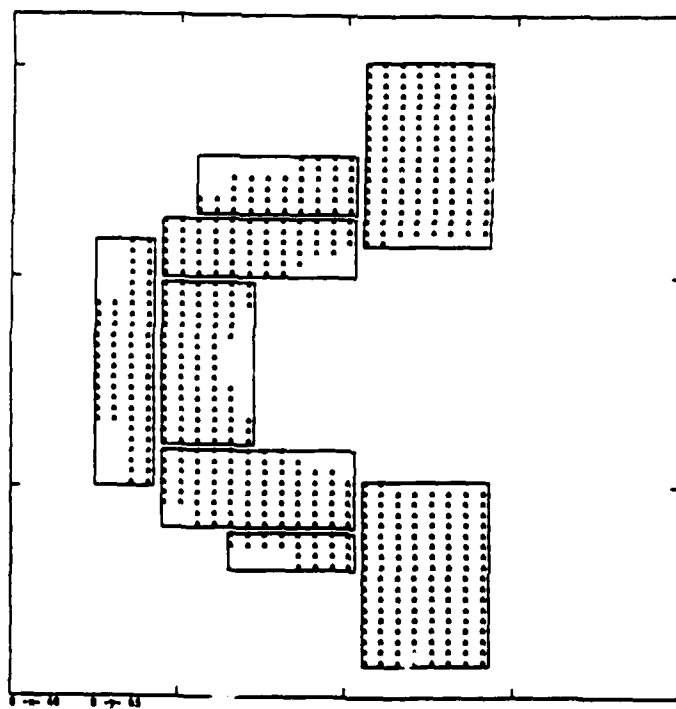
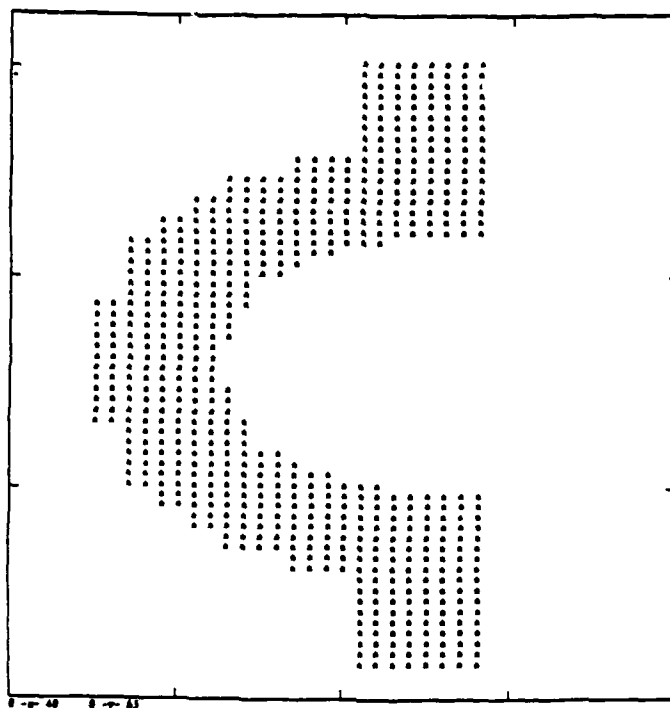


Figure 12 An example of the final algorithm.

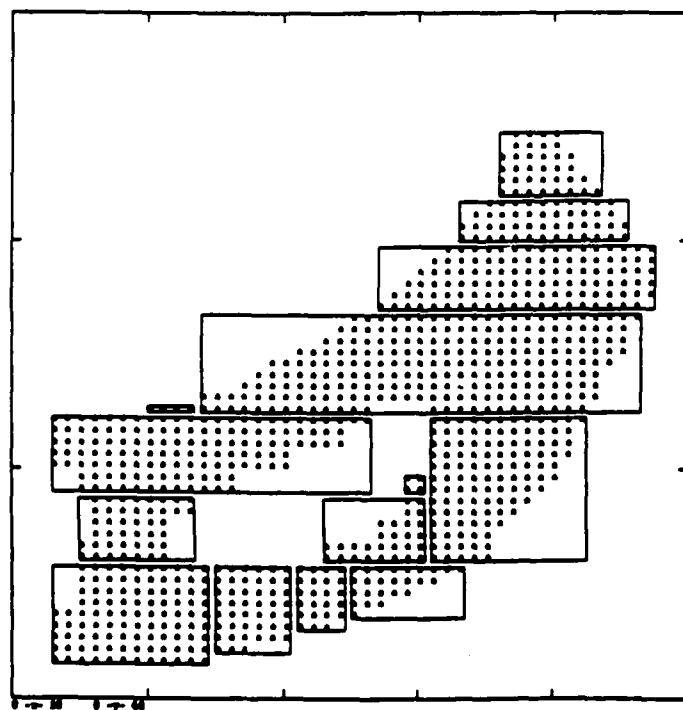
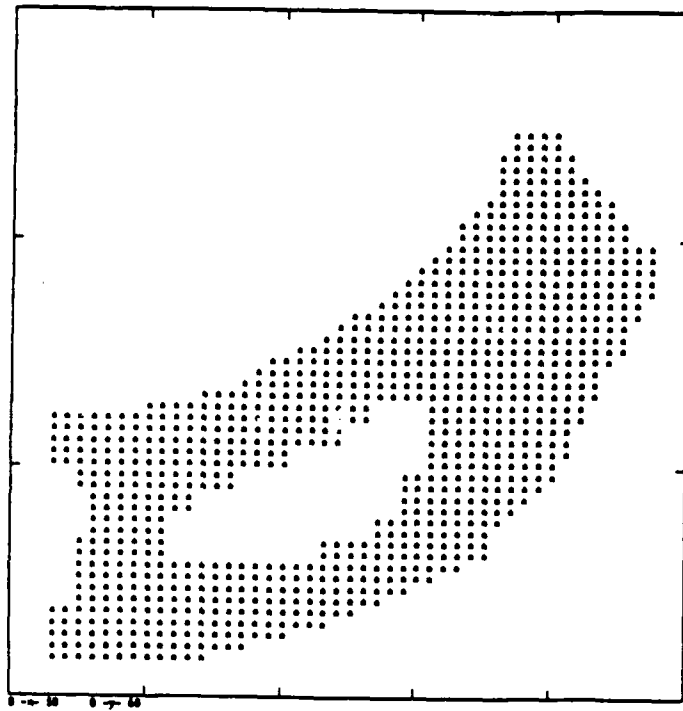


Figure 13 An example of the final algorithm.

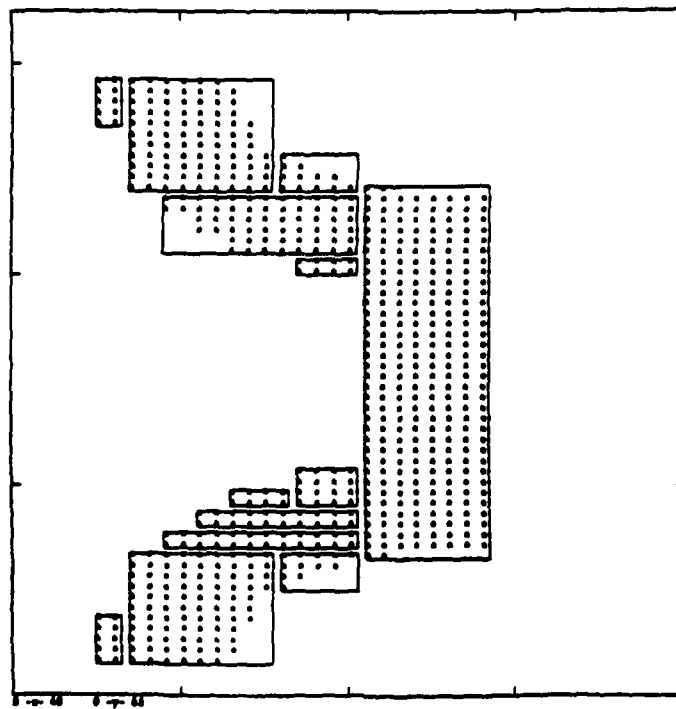
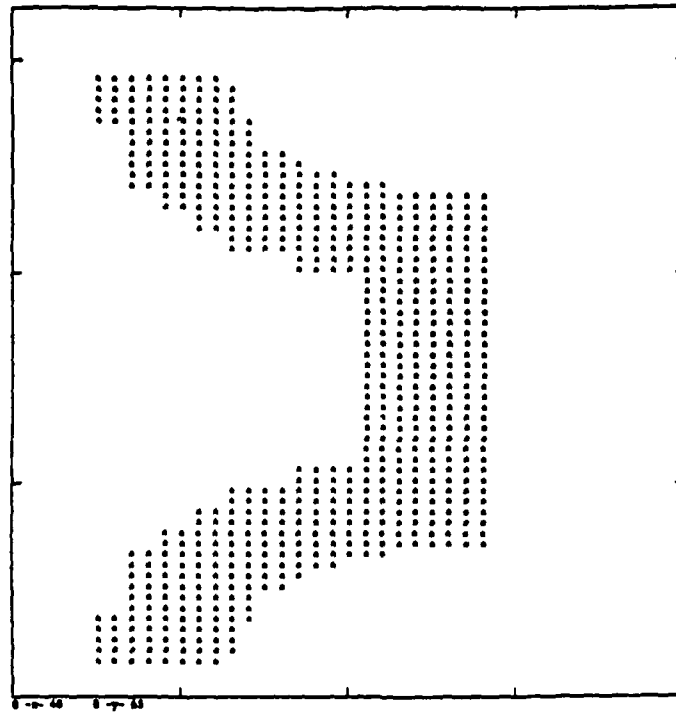


Figure 14 An example of the final algorithm.

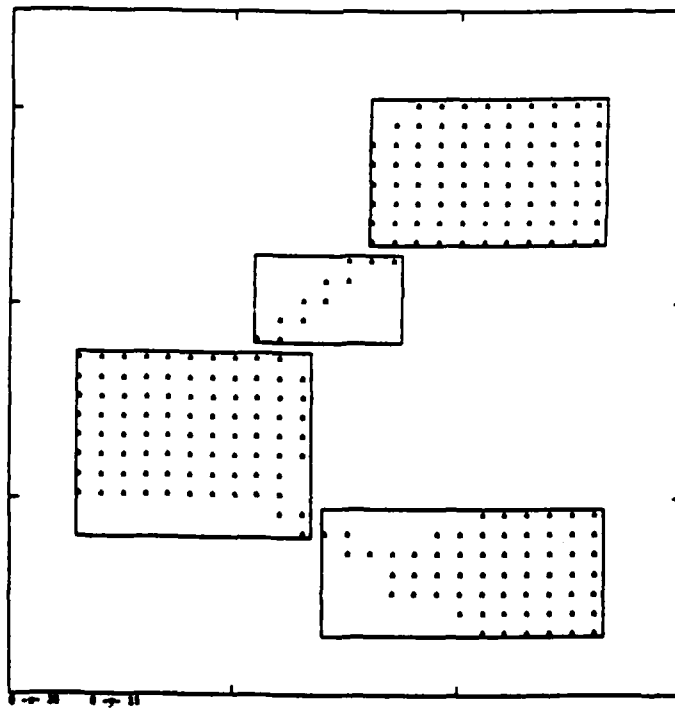
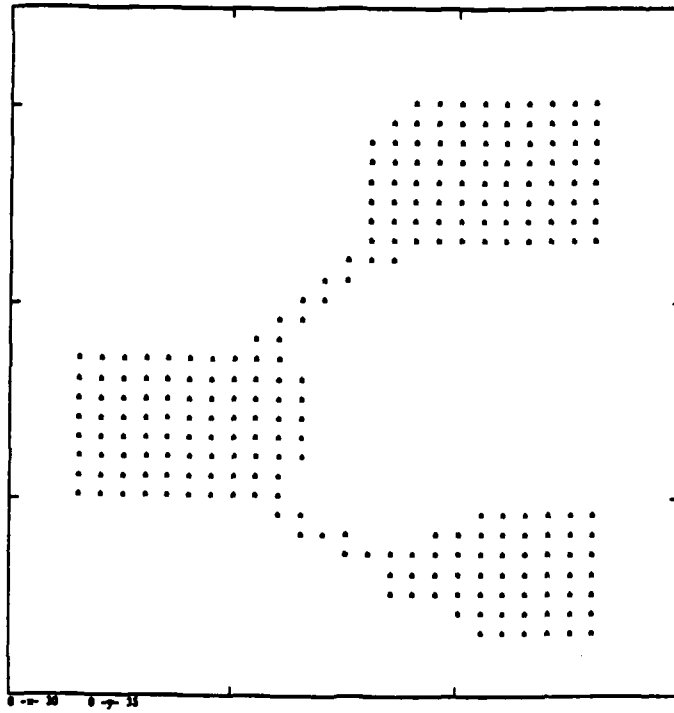


Figure 15 An example of the final algorithm.

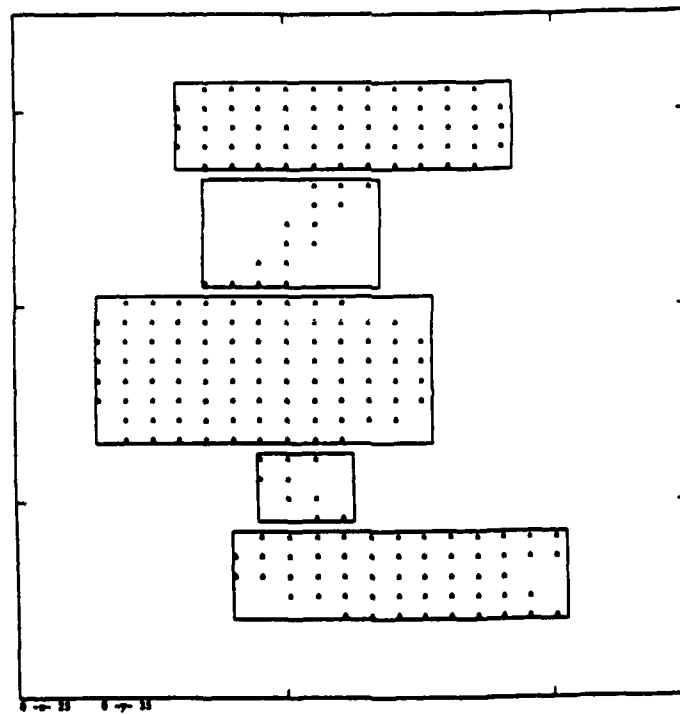
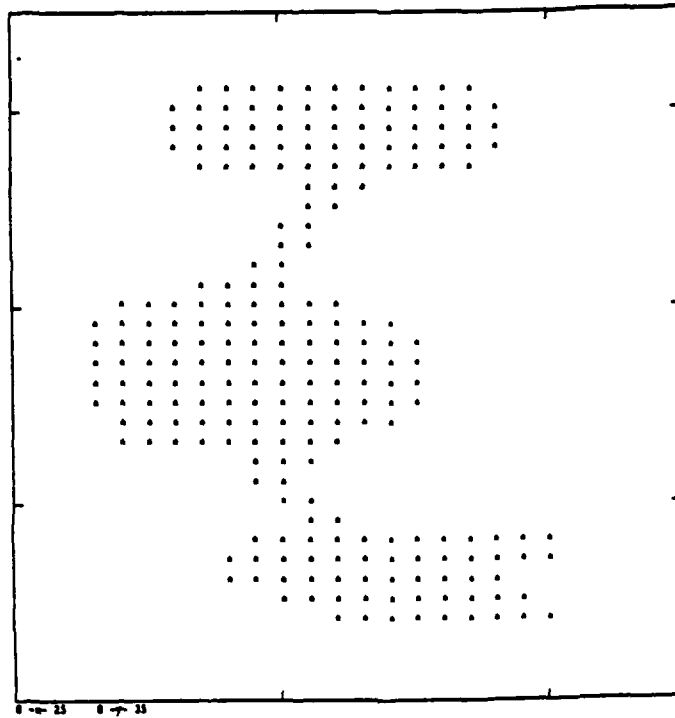


Figure 16 An example of the final algorithm.

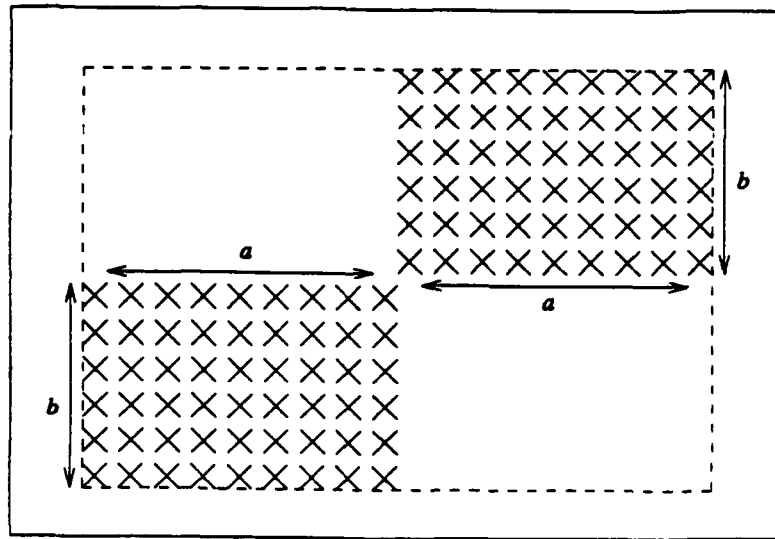


Figure 17 There are no zero crossings in the signature arrays.

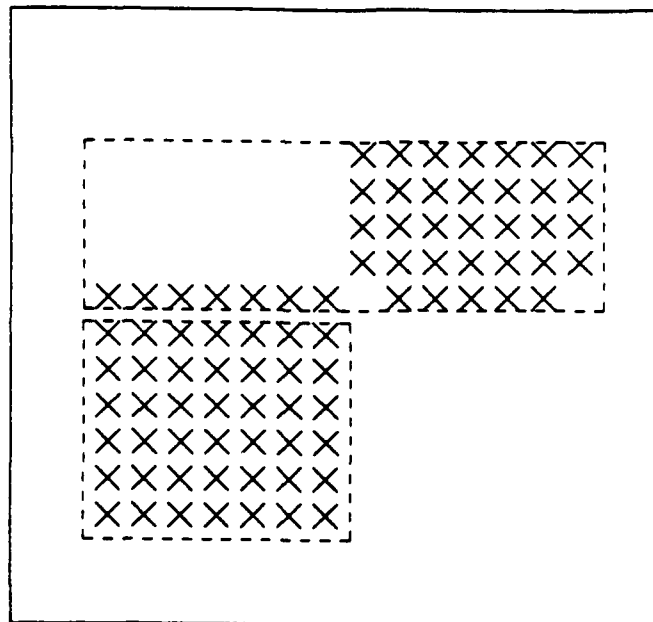


Figure 18 This set of points leads to a non-optimal partition, unless the zero crossings are scaled.

A modification of our algorithm that covers both anomalous cases is to compute the sum of the absolute value of the gradient, and difference the results to get the second derivative. The most robust solution to this problem is still an open question.

A seemingly problematic case is shown in Figure 19. Here it appears as if the algorithm made a non-optimal decision by unnecessarily splitting rectangle R_1 (dotted line) into two smaller subrectangles.

However, careful inspection shows that rectangles $R1$ and $R2$ could not have been generated without introducing an overlapping region.

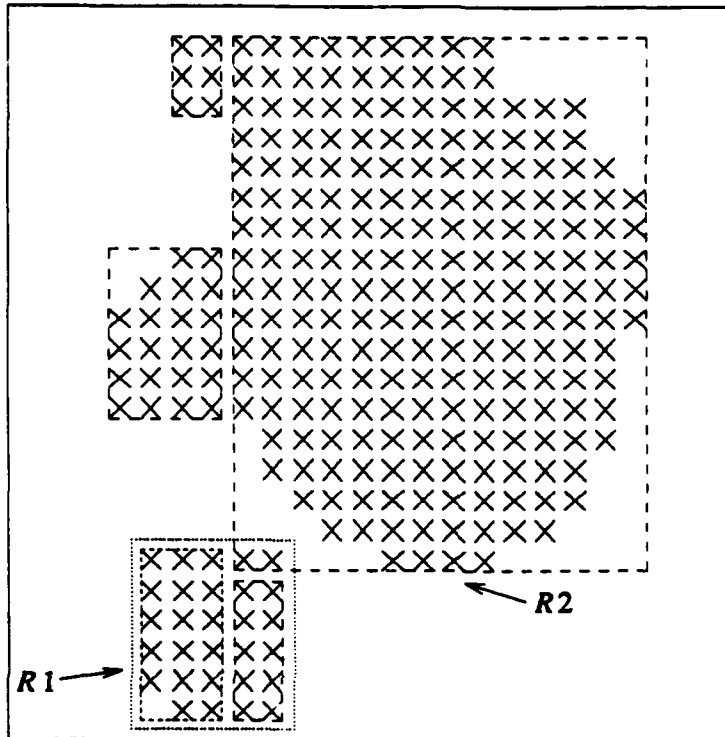


Figure 19 shows an unexpected set of rectangles.

Tight bounds for the running time of the described algorithm are very hard to establish, since the precise flow of the algorithm is input dependent. However, it should be clear that the running time is $O(k(P+N+M))$, where k is the total number of grids upon termination of the algorithm, and P is the number of flagged points. The P term comes from computing the signatures of the points (by traversing a list of the flagged points). The N (resp. M) term comes from the linear search that determines the best inflection point. The above bound is by no means optimal, and the algorithm performs very well in practice. Preliminary three dimensional results also show great performance.

5. Conclusion

We have described a new and efficient algorithm for point clustering and adaptive grid generation. The algorithm's performance has been demonstrated through a series of graphs showing results obtained with both synthetic and actual 2-dimensional inputs. Preliminary experiments with 3-dimensional problems also show a considerably improved performance over the previous approaches. In general, the efficiency of the enclosing rectangles for our applications has been very high, typically ranging between 85% and 100%, with the exception of the problematic cases illustrated in figures 17-19. It is surprising how effective the

algorithm performs on multidimensional data, even though it is based on Cartesian coordinate directions. Finally, this algorithm may also prove useful in other applications with binary image data, for example in generating bounding rectangles for computer graphics applications. A rectangle fitting algorithm has also been used in conjunction with a pattern recognition system for understanding Japanese business cards [Kise et al.].

6. Appendix

6.1 McQueen's k -means Partitioning Algorithm

- [1] Form k single member clusters each one containing precisely one of the k starting seeds. The clusters' centroids originally coincide with the starting seeds.
- [2] Assign each of the remaining data points to the cluster with the nearest (with respect to an appropriate distance metric) centroid recomputing the gaining cluster's centroid after each assignment.
- [3] Assume the cluster centroids are fixed this time, and reassign each of the data points to the cluster with the nearest centroid (one pass through the data).

6.2 McQueen's Converging k -means Partitioning Algorithm

- [1] Form k single member clusters each one containing precisely one of the k starting seeds. The clusters' centroids originally coincide with the starting seeds.
- [2] Assign each of the remaining data points to the cluster with the nearest (with respect to an appropriate distance metric) centroid recomputing the gaining cluster's centroid after each assignment.
- [3] For each data point compute its distance to all the cluster centroids; if the nearest centroid corresponds to a cluster other than the point's actual parent cluster reassign the point; recompute the centroids of both the gaining and losing clusters.
- [4] Repeat step 3) until a full sweep through the data does not induce further changes in the points' memberships.

6.3 Non-Updating Variant of k -means Partitioning Algorithm

- [1] Form k single member clusters each one containing precisely one of the k starting seeds. The clusters' centroids remain fixed throughout the algorithm.
- [2] Assign each of the remaining data points to the cluster with the nearest (with respect to an appropriate distance metric) centroid but do not update the gaining cluster's centroid (one pass through the data).

7. References

- [1] M. Anderberg. *Cluster Analysis for Applications*, Academic Press, 1973.
- [2] D. Ballard and C. Brown. *Computer Vision*. Prentice Hall, 1982.
- [3] M. Berger. Data Structures for Adaptive Grid Generation. SIAM J. Sci. Stat. Comp. 7, July 1986.
- [4] M. Berger and P. Colella. Local Adaptive Mesh Refinement for Shock Hydrodynamics. J. Comp. Phys. 82, May, 1989.
- [5] F.W.C. Campbell and J. Robson. Application of Fourier Analysis to the Visibility of Gratings. Journal of Physiology 197, 1968.
- [6] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.
- [7] J. Hartigan. *Clustering Algorithms*. John Wiley and Sons, 1975.
- [8] B. Horn. *Robot Vision*. MIT Press, 1986.
- [9] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
- [10] K. Kise, K. Yamada, N. Tanaka, N. Babaguchi and Y. Tezuka, Visiting Card Understanding System. Intl. Conf. Pattern Recognition 1988.
- [11] M. Levine. *Vision in Man and Machine*. McGraw-Hill, 1987.
- [12] D. Marr and E. Hildreth. Theory of Edge Detection. Proc. Royal Society of London 207, 1980.
- [13] J. Olinger, Approximate Methods for Atmospheric and Oceanographic Circulation Problems. Lecture Notes in Physics 91, (Glowinski and Lions, eds.), Springer-Verlag, 1979.

STABLE BOUNDARY CONDITIONS FOR CARTESIAN GRID CALCULATIONS

M. J. BERGER† and R. J. LEVEQUE‡

†Courant Institute, 251 Mercer Street, New York University, New York, NY 10012, U.S.A.

‡Department of Mathematics, University of Washington, Seattle, WA 98195, U.S.A.

(Received 20 April 1990)

Abstract—The inviscid Euler equations in complicated geometries are solved using a Cartesian grid. This requires wall boundary conditions in the irregular grid cells near the boundary. Since these cells may be orders of magnitude smaller than the regular grid cells, stability is a primary concern. A new approach to this problem is presented and illustrated.

1. INTRODUCTION

In previous work^{1,3} we have described a Cartesian grid method for the inviscid Euler equations in arbitrary geometries. There are many advantages to be gained from this approach. Grid generation is simplified, since we avoid the use of (possibly multi-block) body-fitted grids, and we can use high resolution, highly efficient solvers on regular grids over the bulk of the domain. This has led to renewed interest in Cartesian grids in recent years (see Refs 4 and 5). One of the difficulties with Cartesian grids is that they give insufficient resolution in certain regions such as leading edges. This can now be overcome by Cartesian adaptive mesh refinement.^{1,6}

The principal remaining difficulty in this approach is due to the essentially arbitrary way that a Cartesian grid intersects the boundaries of the computational domain. In particular, a solid wall boundary cutting through the grid creates irregular cells that may be orders of magnitude smaller than the regular cells away from the boundary. For these irregular cells, special difference equations are needed that maintain stability and accuracy, and satisfy the solid wall boundary conditions of no normal flow.

In this work, we present an improved method for the small boundary cells. We use an explicit, finite volume formulation that computes fluxes at cell edges on the regular part of the domain. We would like to define fluxes at the edges of the irregular cells in such a way that the method is stable even with very small boundary cells, using a time step based on the regular grid cells away from the boundary. The Courant-Friedrichs-Lewy (CFL) condition requires that the numerical method allows information to propagate at least as quickly as the underlying differential equation. In the present context this means that we must define fluxes at the sides of our irregular cells based on more than just the neighboring cell values.

In our previous work, we have used a wave propagation approach in defining these fluxes. Here

we propose an alternative method that has some advantages over the wave propagation approach. In particular, the wave propagation method is subject to intermittent instabilities due to two-dimensional effects that are not clearly understood. The new method has a cancellation property in two dimensions that appears to give better stability properties. Moreover, the computational geometry is simplified in the new approach. The fluxes are defined in terms of weighted averages of nearby cell values. These weights may be calculated as a preprocessing step on any fixed grid and need not be repeatedly calculated. In the previous approach the weights depended on the flow variables and a certain amount of computational geometry was required near the boundary in every time step.

We consider the inviscid Euler equations of gas dynamics in two space dimensions

$$u_t + f(u)_x + g(u)_y = 0, \quad (1)$$

where

$$\begin{aligned} u &= \begin{bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho E \end{bmatrix} \\ f(u) &= \begin{bmatrix} \rho u_1 \\ \rho u_1^2 + p \\ \rho u_1 u_2 \\ u_1(\rho E + p) \end{bmatrix} \\ g(u) &= \begin{bmatrix} \rho u_2 \\ \rho u_1 u_2 \\ \rho u_2^2 + p \\ u_2(\rho E + p) \end{bmatrix}. \end{aligned} \quad (2)$$

Here (u_1, u_2) represents the velocity, E is the total energy per unit mass, and p is the pressure which is

related to the other variables by the equation of state. We assume a γ -law gas, so that

$$p = (\gamma - 1)(\rho E - \frac{1}{2}\rho(u_1^2 + u_2^2)). \quad (3)$$

At a solid wall boundary we require that the component of velocity normal to the wall be zero.

In one space dimension the system reduces to

$$u_t + f(u)_x = 0, \quad (4)$$

where $u = (\rho, \rho v, \rho E)$ and $f(u) = (\rho v, \rho v^2 + p, v(\rho E + p))$, with $v = u_1$ the velocity. The boundary conditions become $v = 0$ at a solid wall.

2. A ONE-DIMENSIONAL EXAMPLE

In order to illustrate this approach we begin with a one-dimensional model problem, the one-dimensional Euler equations for $x > 0$ with a solid wall at $x = 0$. We take a grid with cell interfaces at the points

$$x_0 = 0$$

$$x_1 = h'$$

$$x_j = h' + jh \quad \text{for } j = 2, 3, \dots$$

Here h is a uniform grid spacing and $h' \leq h$. The grid is uniform except for one small cell near the boundary (see Fig. 1). We use a conservative method in the form

$$U_j^{n+1} = U_j^n - \frac{k}{h_j} [F_{j+1}^n - F_j^n], \quad j = 0, 1, \dots \quad (5)$$

Here h_j is the width of the j th cell, so in our case we have $h_0 = h'$ and $h_j = h$ for $j > 0$.

For simplicity we restrict our attention to Godunov's method in the regular portion of the grid, although the ideas we propose can be extended to higher order methods as well. In Godunov's method we take

$$F_j^n = f(u^*(U_{j-1}^n, U_j^n)), \quad (6)$$

where $u^*(u^L, u^R)$ represents the solution to the Riemann problem with left and right states u^L and u^R , evaluated along $x/t = 0$. Although a rigorous stability proof is not available for systems of equations, in practice this method is always stable provided the CFL condition

$$\left| \frac{k \lambda_{\max}}{h} \right| \leq 1 \quad (7)$$

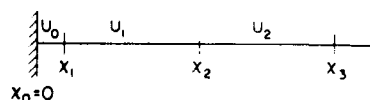


Fig. 1. One-dimensional grid with one irregular cell adjacent to the wall.

is satisfied, where λ_{\max} is the maximum wave speed. We will assume that our time step k is chosen so that the condition (7) is satisfied relative to the uniform h . We will use the flux Eq. (6) for $j = 2, 3, \dots$, i.e. at all interfaces where the cell on both sides is regular. Our task is to define fluxes F_j^n for $j = 0, 1$ so that we maintain stability (and accuracy) with this time step even if $h' \ll h$.

First suppose $h' = h$. Then we can use the Godunov flux [Eq. (6)] also at $j = 1$. At the wall we use the well-known observation that the solution to the boundary value problem can be obtained by ignoring the wall and extending the computational domain to the whole line $-\infty < x < \infty$, if we take data $u_0(x)$ for $x < 0$ equal to

$$\rho(x, 0) = \rho(-x, 0)$$

$$v(x, 0) = -v(-x, 0) \quad \text{for } x < 0$$

$$p(x, 0) = p(-x, 0).$$

We will denote this "reflection" of the data (in which the velocity is negated) by the operator \mathcal{R} , so that in shorthand we can write

$$u(x, 0) = \mathcal{R}(u(-x, 0)) \quad \text{for } x < 0.$$

With this extended data, the solution continues to satisfy $u(x, t) = \mathcal{R}(u(-x, t))$ also for $t > 0$ and in particular the boundary condition $u(0, t) = 0$ is automatically satisfied. This suggests that we obtain a flux at the wall by solving a Riemann problem with left and right states

$$u^L = \mathcal{R}(U_0) \quad u^R = U_0$$

in each time step to obtain

$$F_0 = f(u^*(\mathcal{R}(U_0), U_0)).$$

(For brevity we will leave off the superscript n in general.) Note that the density and energy components of this flux will be zero since the velocity component u^* is zero at the wall. There will only be a momentum flux at the wall due to the pressure there, as expected physically.

If $h' < h$ we could attempt to use this formula to define F_0 but we would find that it is unstable unless the CFL condition

$$\left| \frac{k \lambda_{\max}}{h'} \right| \leq 1 \quad (8)$$

is satisfied. This will place an unreasonable restriction on k if $h' \ll h$.

This instability is caused by the fact that the boundary flux F_0 is based on the data U_0 alone. If the CFL condition (8) is satisfied, then it is only this data that affects the flux at the wall over the time step

However, when Eq. (8) is violated the value U_1 should also affect the flux at the wall, and ignoring this effect leads to instability.

In a "large time step" approach we increase the stencil of the method, meaning we allow more data points to come into the computation of each flux, and hence retain stability. One way to achieve this is by a wave propagation approach. The solution of the Riemann problem at each cell interface consists of three waves propagating away from the interface. If Eq. (8) is satisfied then these waves remain in the cells bordering the interface during the entire time step and hence affect the solution only in these cells. If Eq. (8) is violated then the waves may affect cells further away. Implementing Godunov's method in terms of this wave propagation approach, allowing waves to affect more than just the adjacent cell, gives a large time step generalization that remains stable for much larger time steps.⁷ In the present context this allows us to reduce h' without reducing the time step k . Waves from the boundary Riemann problem cross the interface at x_1 and affect U_1 as well as U_0 . Waves from the interface at x_1 may reach the boundary. These waves reflect off the boundary and the reflected wave affects the value U_0 and perhaps also U_1 if the reflected wave reaches the cell interface at x_1 during the time step.

A more detailed description of this procedure may be found in Ref. 3. A natural extension to two space dimensions gives one method to deal with small cells near the boundary, as described in Refs 1-3. In one dimension this works very well but in two dimensions occasional stability problems have still been observed due to multidimensional effects.

2.1. The new approach

Our new approach to the small cell problem can also be illustrated with the one-dimensional problem described above. We again use the method of Eq. (5) with Godunov fluxes [Eq. (6)] for $j = 2, 3, \dots$. For $j = 0$ and $j = 1$ we define fluxes in a similar manner but with a different choice of states u^L and u^R in the Riemann solver. Recall that in a naive attempt to use Godunov's method regardless of the size h' of the small cell we would take left and right states

$$u_0^L = \mathcal{A}(U_0) \quad u_0^R = U_0 \quad (9)$$

$$u_1^L = U_0 \quad u_1^R = U_1. \quad (10)$$

To maintain stability when h' is small, we need to allow data from additional grid cells to affect the left and right states at each of these interfaces. Recall that the method is assumed to be stable with our choice of k and h on the regular portion of the grid. This suggests that we would define u_1^L by taking the average value of U over an interval of length h to the left of the interface x_1 and define u_0^R by taking the average value of U over an interval of length h to the right of x_1 .

For example, at $x_0 = 0$ (the wall) we set

$$u_0^R = \frac{1}{h} (h' U_0 + (h - h') U_1). \quad (11)$$

If we view the grid values as defining a piecewise constant function with values U_j in the j th cell, then Eq. (11) is the average value of this function over the interval $0 \leq x \leq h$. Note that if $h' = h$ (the grid is completely regular) then Eq. (11) reduces to $u_0^R = U_0$ as expected for Godunov's method. Recall that in Godunov's method we take $u_0^L = \mathcal{A}(U_0) = \mathcal{A}(u_0^R)$ to impose the boundary condition $v(0, t) = 0$. This suggests that more generally we take

$$u_0^L = \mathcal{A}(u_0^R), \quad (12)$$

where u_0^R is defined by Eq. (11). We then use the Godunov flux

$$F_0 = f(u^*(u_0^L, u_0^R)) \quad (13)$$

as the flux at the wall. Using Eq. (12) guarantees that there will be no flux of mass or energy through the wall and hence that the method is conservative.

To define the left and right states of x_1 we again construct intervals of length h to either side of this point and average the piecewise constant function defined by U over these intervals. To the right of x_1 lies a regular cell of length h , and so

$$u_1^R = U_1. \quad (14)$$

To the left of x_1 an interval of length h extends beyond the wall (assuming $h' < h$). Beyond the wall we assume that U takes the value u_0^L given by Eq. (12). A weighted average of this value and U_0 gives u_1^L

$$u_1^L = \frac{1}{h} (h' U_0 + (h - h') u_0^L). \quad (15)$$

The flux f_1 is then defined by

$$F_1 = f(u^*(u_1^L, u_1^R)). \quad (16)$$

Again, if $h' = h$ this reduces to the standard Godunov flux.

This method remains stable even when $h' \ll h$. To see why this should be so, consider the formula (5) for $j = 0$ where $h_1 = h'$. It is the division by h' that may cause stability problems unless the fluxes F_0 and F_1 themselves agree to $O(h')$ as $h' \rightarrow 0$. The Godunov fluxes based on Eqs (9) and (10) do not have this property. However, our proposed fluxes (13) and (16) do have this property, since inspection of the formulas (11), (12), (14) and (15) shows that $u_1^L = u_0^L + O(h')$ and $u_0^R = u_1^R + O(h')$ as $h' \rightarrow 0$. Since the flux function $f(u^*(u^L, u^R))$ is a Lipschitz continuous function of u^L and u^R , it follows that $F_1 - F_0 = O(h')$ as $h' \rightarrow 0$ and there is at least a chance that the method remains

stable for arbitrary $h' \ll h$. Numerical experiments show that this is indeed the case (although it is possible to contrive examples, such as a strong rarefaction wave originating at this irregularity, where the results are not very accurate).

3. BOUNDARY CONDITIONS IN TWO DIMENSIONS

Turning now to the two-dimensional problem, we will give a brief description of how the idea described above extends to handle the small cell problem.

Consider the portion of the boundary shown in Fig. 2a and a typical boundary cell (i, j) . The formula for updating the value U_{ij} is the two-dimensional analog of Eq. (5)

$$U_{ij}^{n+1} = U_{ij} - \frac{k}{A_{ij}} [F_{i+1,j} - F_{ij} + G_{i,j+1} - G_{ij} + H_{ij}]. \quad (17)$$

The fluxes F , G , and H represent flux per unit time through the corresponding side of the grid cell (see Fig. 2b) and A_{ij} is the area of the cell. If any of the sides are missing, the corresponding flux is zero.

On regular grid cells, $H_{ij} = 0$ and the fluxes F and G might be defined by an extension of the Godunov method, setting

$$\begin{aligned} F_{ij} &= hf(u^*(U_{i-1,j}, U_{ij})) \\ G_{ij} &= hg(u^*(U_{i,j-1}, U_{ij})). \end{aligned} \quad (18)$$

Here u^* represents the solution to the appropriate Riemann problem in the x or y direction. Note that the fluxes include the factor h , the length of each side, to give a flux per unit time across the side.

It is the denominator A_{ij} in Eq. (17) that causes trouble when the cell is very small. We again assume the method is stable on the regular portion of the grid, where $A_{ij} = h^2$. To maintain stability we need to insure that our formulas for the fluxes cause the total flux [the sum in brackets in Eq. (17)] to cancel to $O(A_{ij})$ as $A_{ij} \rightarrow 0$. This is only possible if the fluxes are computed via formulas that involve more than just the two cells bordering the cell side. We take an approach analogous to what we described above in one dimension.

3.1. Boundary fluxes

We begin by considering the boundary segment, where we must compute the flux H_{ij} . In two dimensions

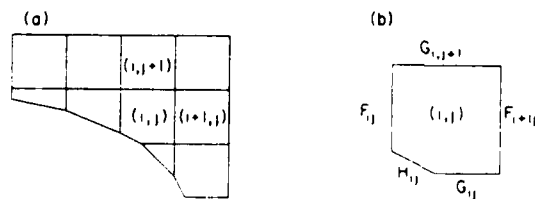


Fig. 2. (a) The Cartesian grid near the boundary. (b) Blow-up of cell (i, j) showing the location of fluxes.

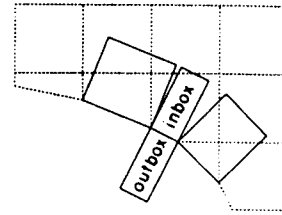


Fig. 3. The inbox and outbox constructed from the boundary segment of cell (i, j) , and the inbox for two neighboring cells.

sions the solid wall boundary condition requires that the normal velocity at the wall be equal to zero. If we have some value u_{ij}^{in} representing the value of u just inside the wall, then we can obtain the flux H_{ij} by solving a one-dimensional Riemann problem in the direction normal to the wall, with left and right states

$$u_{ij}^L = \mathcal{R}(u_{ij}^{\text{in}}) \quad u_{ij}^R = u_{ij}^{\text{in}}.$$

The reflection operator \mathcal{R} is now defined by negating the normal velocity component while leaving the tangential velocity component along with the density and pressure unchanged. The resulting Godunov flux is used for H_{ij} .

We obtain u_{ij}^{in} by a procedure analogous to that of the one-dimensional example. We construct a box extending a distance h away from the wall as shown in Fig. 3. The box extending into the computational domain is called $\text{inbox}(i, j)$. The mirror image box outside the domain is called $\text{outbox}(i, j)$. We obtain the value u_{ij}^{in} by viewing the given data U as defining a piecewise constant function, constant in each grid cell, and setting u_{ij}^{in} to be the average value of this function over the region $\text{inbox}(i, j)$. In Fig. 3 $\text{inbox}(i, j)$ would contain an area-weighted average of two grid values while the value for $\text{inbox}(i-1, j)$ is based on four grid values. We think of the outbox as containing the value $u_{ij}^{\text{out}} = \mathcal{R}(u_{ij}^{\text{in}})$.

To find the weights needed to compute u_{ij}^{in} we must compute the intersection of the inbox with each nearby cell. This is easily accomplished with standard computational geometry routines. Note that for a given geometry and grid these weights need only be computed once at the beginning of the computation. They need not be recomputed in each time step.

3.2. Fluxes at other sides

We now consider the fluxes F and G along other sides of this cell. These are all computed by similar procedures, so to be specific we will consider the computation of F_{ij} , the flux on the left side of this cell.

To compute F_{ij} we solve two Riemann problems, one in some direction ξ with data u_{ij}^L, u_{ij}^R and the other in the orthogonal direction η with data u_{ij}^L, u_{ij}^R . The choice of these directions and the data will be discussed in a moment. First we explain how these Riemann problem solutions are computed and used to define F_{ij} .

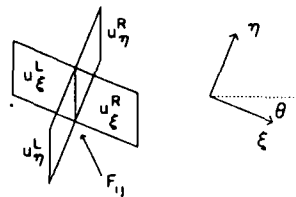


Fig. 4. A vertical cell interface and the ξ - and η -directions.

Figure 4 shows a typical vertical cell interface and two orthogonal directions ξ and η . Let θ be the angle that ξ is rotated from the x -direction ($\theta < 0$ in this example). Suppose we solve a one-dimensional Riemann problem in the ξ -direction with left and right states u_ξ^L, u_ξ^R to obtain the flux per unit length per unit time in the ξ -direction. (To do this we rotate the velocity components of u_ξ^L, u_ξ^R into ξ - η velocity components, solve the one-dimensional Riemann problem, and then rotate the resulting flux f back to x - y velocity components.) Call this resulting flux f_ξ .

Similarly, solving a one-dimensional Riemann problem in the η -direction with left and right states u_η^L, u_η^R gives f_η , the flux per unit length per unit time in the η -direction. The total flux across the vertical segment of length h is then

$$F = h'(f_\xi \cos \theta - f_\eta \sin \theta). \quad (19)$$

This is the value we use for the flux F_y .

This same approach has been used by others (see Refs 8-10) to define multidimensional upwind methods. In these methods the directions ξ and η are chosen based on the local flow in an attempt to use physically meaningful directions in place of the artificial coordinate directions. For example, the direction of the velocity or the pressure gradient might be used to define ξ . In our application we are only considering cells adjacent to the boundary and the relevant directions are the directions tangential and normal to the wall. We choose ξ to be the direction tangential to the wall in one of the two cells bordering this interface. Since our primary concern is to maintain stability in very small cells, we choose the smaller of the two adjacent cells to define this direction. This will lead to cancellation of fluxes in tiny cells in the same manner as previously seen in the one-dimensional example. The η -direction is normal to the ξ -direction.

3.3. Tangential boxes

We must still specify the data for these tangential and normal Riemann problems. We first consider the tangential problem. We use an approach similar to the specification of data in an inbox described above. From the interface we construct boxes that extend a distance h in the ξ -direction. Figure 5a shows an example. The data u_ξ^L, u_ξ^R are obtained by an area-weighted average of the values in each cell the box

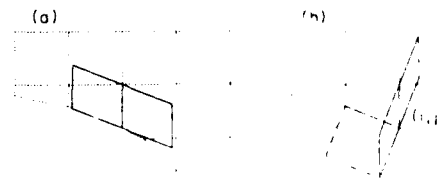


Fig. 5. (a) Tangential boxes constructed from the cell interface. (b) Normal boxes from the cell interface and outbox($i-1, j$).

overlaps. In our current implementation we assume the wall is convex, so that these boxes lie entirely within the computational domain. Each box overlaps at most two grid cells and the weights are easily calculated. Since the directions ξ and η and the resulting boxes depend only on the geometry, not on the flow variables, these weights can again be calculated once and for all as a preprocessing step.

3.4. Normal boxes

Figure 5b shows the normal boxes in the η -direction. The box in the outward direction does not hit the boundary and overlaps at most two regular cells, so u_η^R is calculated as an area-weighted average of these cell values. The other box may extend beyond the boundary. If so, the portion lying outside the computational domain lies in one or more outboxes, the artificial cells created in the process of computing the boundary flux H_y described above. Figure 5b shows a simple example where the normal box intersects only one cell ($i-1, j$) and outbox($i-1, j$). More generally the normal box might intersect two cells and their outboxes, as happens for example when we compute the flux $F_{i+1, j}$ which involves cells (i, j) and ($i, j+1$). Moreover the two outboxes will in general overlap due to the convexity of the region. We again use area-weighted averaging over the four cells in question, weighting the values $U, U_{i, j-1}, u_\eta^{out}, u_{i, j-1}^{out}$ by the areas of intersection and then dividing by the sum of all these areas.

3.5. Cancellation

Although we will not present the details here, it can be shown that this way of defining fluxes leads to the desired cancellation of fluxes in very small cells. Let values of u_ξ^L computed at each of the three sides of a very small triangular cell are nearly the same because of our construction. They differ by only $O(A_{ij})$ as $A_{ij} \rightarrow 0$. The same is true of each of the other values $u_\xi^R, u_\eta^L, u_\eta^R$ and so by Lipschitz continuity of the fluxes F, G and H we obtain the required cancellation. Numerical results show stability even when A_{ij} is many orders of magnitude less than h^2 .

3.6. Higher order methods

The method (17) with fluxes (18) is only first order accurate and is highly dissipative. In our previous work we used the wave propagation boundary conditions together with a high resolution method away

from the boundary and obtained reasonable results (see Ref. 1). The new boundary conditions can also be applied in conjunction with a high resolution method and give similar results. Moreover, with our new formulation it appears to be easier to improve the accuracy of the boundary conditions, allowing us to obtain higher order accuracy overall. The main idea is to introduce slopes in each cell and use piecewise linear approximations in place of piecewise constants to define the fluxes. Near the boundary we can easily estimate slopes in the tangential direction along the wall by differencing values in the inboxes that we have defined above.

These improvements are still being investigated and will be reported in detail elsewhere. Here we will only compare results obtained with the method as we have described it and results obtained using the same interior method with the wave propagation boundary conditions described in earlier papers.

4. NUMERICAL RESULTS

We show one representative test case, a supersonic shock going around an expansion corner. We also show the steady state solution obtained at large times. The exact rarefaction wave solution is a simple wave and can be computed following Sec. 6.17 of Whitham,¹¹ for example.

The geometry we use is the rectangle $[0, 1.32] \times [0, 0.8]$ with a solid wall at

$$y = \begin{cases} 0.3 & x \leq 0.1 \\ 0.3(1 - (x - 0.1)^2) & 0.1 \leq x \leq 0.7 \\ 0.192 - 0.36(x - 0.7) & 0.7 \leq x \leq 1.32. \end{cases}$$

The initial conditions consist of a Mach 2.31 shock at $x = 0.06$ with left and right states

$$\rho^L = 5.1432, \quad u_1^L = 2.04511, \quad u_2^L = 0,$$

$$p^L = 9.04545$$

and

$$\rho^R = 1.4, \quad u_1^R = 0, \quad u_2^R = 0, \quad p^R = 1.0.$$

We take $h = 0.02$ (66×40 grid) and a time step $k = 0.002$. This corresponds to a Courant number of roughly 0.37 relative to the regular cells with area h^2 . For the crude form of Godunov's method used here, the stability restriction requires a Courant number of less than 0.5. The smallest cells near the boundary have an area of roughly $10^{-3} h^2$.

Figure 6 shows numerical results at time $t = 0.4$, as the shock is rounding the corner. Results obtained with the wave propagation boundary conditions are shown in Fig. 6a, while Fig. 6b shows the results obtained with our new approach. These results are very similar. Slight discrepancies can be seen near the wall just around the shock. For this problem, both sets of boundary conditions worked well. We have also performed tests on other problems where the wave propagation method shows instabilities and have observed no such difficulties with the new method.

Figure 7 shows the steady state results obtained after many iterations of the time dependent code (no attempt has been made so far to accelerate convergence for steady state solutions). We only show the results with our new boundary conditions. The wave

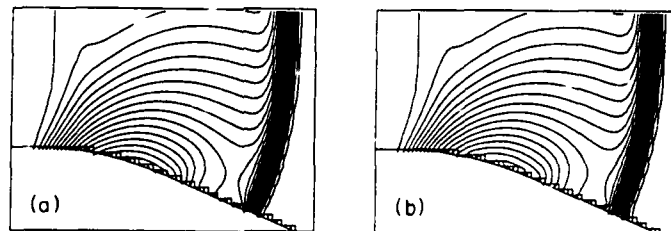


Fig. 6. Shock propagation results at $t = 0.4$. (a) Using the wave propagation boundary conditions. (b) Using the new boundary conditions.

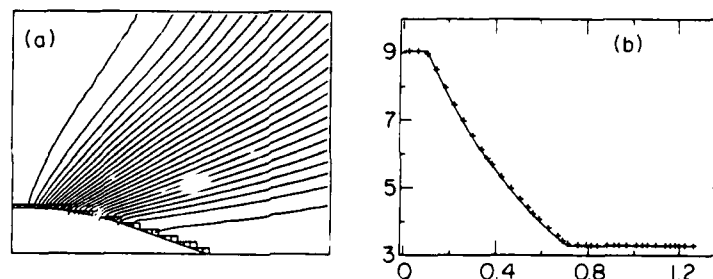


Fig. 7. Steady state results. (a) Pressure contours. (b) Pressure along the wall. The solid line is the exact solution. + indicates the numerical solution.

propagation boundary conditions give very similar results. We use a coarser grid than in the previous example ($h = 0.04$) in order to demonstrate that we achieve reasonable accuracy along the boundary even with a relatively coarse piecewise linear representation of the boundary. We also use a larger computational domain, $[0, 2] \times [0, 1.6]$, to minimize the impact of the far-field boundaries. The true solution is a rarefaction wave originating from the portion of the boundary with nonzero curvature. In the exact solution the contour lines would be straight lines. Our results are contaminated by effects from the far-field boundary.

Near the solid wall the contour lines appear to show a boundary layer. This is an artifact of the graphics routine, which assumes the data are on a uniform grid at cell centers. Our data near the boundary should be viewed as an approximation to the pointwise value at the center of mass of the irregular cell, not at the center of the full Cartesian cell.

In order to examine the accuracy at the wall, Fig. 7b shows plots of the pressure along the boundary, plotted against arclength. To obtain a boundary pressure, the cell value U_{ij} and the reflected value $\mathcal{R}(U_{ij})$ are used to solve the one-dimensional Riemann problem normal to the wall in each irregular cell. The resulting pressure p^* is used as the boundary pressure. Figure 7b shows these results and also the exact solution (to machine precision) calculated using the theory of Ref. 11.

In more complicated computations we use adaptive grid refinement to obtain high resolution results with minimal effort. The boundary conditions described here can also be used in conjunction with the adaptive Cartesian grid code described in Refs 1 and 2.

Acknowledgments—It is a pleasure to acknowledge several stimulating conversations with John Bell and Phil Colella.

The authors were supported in part by NSF Grants ASC-8858101 and DMS-8657319, AFOSR Grant 86-0148, and DOE Grant DE-FG02-88ER25053. This work was also supported in part by NASA Contract NAS1-18605 while the authors were in residence at ICASE, NASA Langley Research Center.

REFERENCES

1. M. Berger and R. J. LeVeque, "An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries," AIAA paper AIAA-89-1930, 1989.
2. R. J. LeVeque, "Cartesian grid methods for flow in irregular regions," in *Numerical Methods in Fluid Dynamics III* (edited by K. W. Morton and M. J. Baines), pp. 375-382, Clarendon Press, Oxford.
3. R. J. LeVeque, "High resolution finite volume methods on arbitrary grids via wave propagation," *Journal of Computational Physics* **78**, 36-63 (1988).
4. H. H. D. Clarke and M. Salas, "Euler calculations for multielement airfoils using Cartesian grids," AIAA Paper 85-0291, 1985.
5. B. Wedan and J. South, "A method for solving the transonic full-potential equations for general configurations," Proc. AIAA Computational Fluid Dynamics Conference, 1983.
6. M. J. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," *Journal of Computational Physics* **82**, 64-84 (1989).
7. R. J. LeVeque, "A large time step generalization of Godunov's method for systems of conservation laws," *SIAM Journal of Numerical Analysis* **22**, 1051-1073 (1985).
8. S. F. Davis, "A rotationally biased upwind difference scheme for the Euler equations," *Journal of Computational Physics* **56**, 65-92 (1984).
9. D. W. Levy, K. G. Powell and B. van Leer, "An implementation of a grid-independent upwind scheme for the Euler equations," AIAA Paper 89-1931-CP, 1989.
10. K. G. Powell and B. van Leer, "A genuinely multi-dimensional upwind cell-vertex scheme for the Euler equations," AIAA Paper 89-0095, Reno, 1989.
11. G. Whitham, *Linear and Nonlinear Waves*, Wiley Interscience, New York, 1974.

Local Adaptive Mesh Refinement for Shock Hydrodynamics

M. J. BERGER

*Courant Institute of Mathematical Sciences, New York University,
251 Mercer Street, New York, 10012 New York*

AND

P. COLELLA

Lawrence Livermore Laboratory, Livermore, 94550 California

Received September 8, 1987; revised May 20, 1988

The aim of this work is the development of an automatic, adaptive mesh refinement strategy for solving hyperbolic conservation laws in two dimensions. There are two main difficulties in doing this. The first problem is due to the presence of discontinuities in the solution and the effect on them of discontinuities in the mesh. The second problem is how to organize the algorithm to minimize memory and CPU overhead. This is an important consideration and will continue to be important as more sophisticated algorithms that use data structures other than arrays are developed for use on vector and parallel computers. © 1989 Academic Press, Inc.

1. INTRODUCTION

In this paper, we present computations that use adaptive mesh refinement to solve multidimensional, time dependent shock hydrodynamics problems. Complicated structures such as multiple Mach reflections arise in these problems. Adaptive techniques are essential for our computations in order to adequately resolve features in the solution within today's computer limitations.

Our starting point will be the algorithms in [6] for adaptive mesh refinement for hyperbolic equations on rectangular grids. In this approach, the refined regions consist of a small number of rectangular grid patches with finer mesh spacing than the underlying global coarse grid. These rectangular subgrids contain points where the error in the coarser grid solution is too high, and other points as well. We use rectangular subgrids so that we can use integration methods for rectangular grids whose convergence properties are well understood. These methods can be made quite efficient on vector and parallel computers. In addition, rectangular grids have a simple user interface. We can use the same integrator on fine and coarse grids. By separating the integrator from the adaptive strategy, an off-the-shelf integrator can

be used without modification. This eliminates much of the problem specific work in doing adaptive calculations.

The present work differs from that in [6] in several respects. The main one is that we are computing unsteady flows with shocks, so that maintaining global conservation form is a primary consideration. The second difference is that the nested refinements we use have boundaries coinciding with the grid lines of the underlying coarse mesh. This greatly simplifies the maintenance of conservation over the former approach, where the refined subgrids were allowed to be rotated with respect to the coarse grid. Third, great care was taken to obtain an efficient implementation on a supercomputer. The main difficulty with adaptive methods is the need for data structures not usually found in numerical software. We felt the program complexity was high enough to justify the effort of devising as general and automatic an approach as possible.

Earlier work along the lines of the present work was done by [7] in one dimension, and [14] for scalar problems in two dimensions. [19] have also computed transonic flow in two dimensions with grid embedding. However, in the latter two approaches, the grids were not restricted to rectangles. The data structures, and therefore the efficiency of such an approach, are quite different. Our method of adaptivity through grid refinement is in contrast to methods that adapt the grid by moving grid lines into one region, leaving a coarser region somewhere else [18, 15, 13, 20, 8]. Such methods try to get the most accurate solution for a fixed cost, whereas our approach tries to attain a fixed accuracy for a minimum cost. Both approaches have their advantages and disadvantages. The so-called moving grid point methods often have trouble maintaining a smooth grid. Regularity terms and penalty functions used to regularize the grid add overhead and reduce the simplicity of these methods. Local grid refinement, on the other hand, has the drawback of needing special equations at grid interfaces. In a method where a fixed number of grid points are used during a computation, the user must initially guess at what will be an adequate number of points to resolve features in the solution that may arise later. With local grid refinement, grid points are added or removed as necessary.

In the numerical experiments shown below, we have combined this adaptive mesh refinement strategy with the high resolution difference scheme of [10] to develop an almost automatic software tool for solving gas dynamics problems in two space dimensions. A reasonable question is, why is an adaptive method needed, given that the difference scheme used, a second-order Godunov-type method, already has quite high resolution? Conversely, if adaptive methods are used, are such complicated and expensive difference schemes really necessary? The answer is that both components are necessary to obtain well-resolved results for shock hydrodynamics. It has been demonstrated [22] that the more complicated Godunov-type schemes give more resolution per computational dollar than simpler schemes such as Lax-Wendroff. Given that a high quality scheme is necessary, adaptive mesh refinement can then concentrate the computational effort in regions where it is most useful. Since Godunov-type methods are more expensive than simple schemes, the computational savings of selective refinement can be substan-

tial. Some of the computations presented here could not have been done reasonably without the use of an adaptive solver.

In the sections that follow, we describe the adaptive mesh refinement (AMR) algorithm for integrating a general hyperbolic system of conservation laws

$$\begin{aligned} u_t + f(u)_x + g(u)_y &= 0 & \text{on } D \subset \mathbb{R}^2 \\ Bu &= b & \text{on } \partial D. \end{aligned}$$

Our numerical examples involve the Euler equations for gas dynamics, where

$$u = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \quad f(u) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uE + up \end{pmatrix}, \quad g(u) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vE + vp \end{pmatrix}.$$

and

$$p = (\gamma - 1) \left(\rho E - \rho \left(\frac{u^2 + v^2}{2} \right) \right).$$

Although our work to date is in two space dimensions, all the algorithms extend to three dimensions, and in fact it seems possible to implement a general code where the number of dimensions is input.

In the next sections we describe in detail the adaptive mesh refinement algorithm and its implementation. We give enough detail for users interested in modifying the algorithm, using our code, or writing their own. First, we discuss the structures that define our grid hierarchy. Next, we describe the integration scheme for such a (static) grid hierarchy. Third, the grid generation and error estimation procedures used to generate the grid hierarchy itself are presented. Our error estimation procedure is theoretically justifiable only for smooth solutions. We discuss variations of it that may prove useful for problems with shocks. In the last section we present numerical experiments along with a detailed timing analysis of the runs. This program is being used to study Mach reflection in two dimensions with resolution not previously possible. New results, a triple Mach stem configuration at low γ , have been observed.

2. GRID DESCRIPTION

AMR is based on using a sequence of nested, logically rectangular meshes on which the pde is discretized. In this work, we require the domain D to be a finite union of rectangles whose sides lie in the coordinate directions. We assume here that all the meshes are physically rectangular as well, although this is not essential. The method discussed here can be implemented on a general quadrilateral mesh.

(See, for example, [5]). We define a sequence of levels $l = 1, \dots, l_{\max}$. A grid $G_{l,k}$ has mesh spacing h_l , with level 1 coarsest, and define

$$G_l = \bigcup_k G_{l,k}.$$

With an abuse of terminology describing a grid and the domain it covers, we have $G_1 = \emptyset$, $G_{l_{\max}} = D$, the problem domain. If there are several grids at level 1, the grid lines must "align" with each other; that is, each grid is a subset of a rectangular discretization of the whole space.

We may often have overlapping grids at the same level, so that $G_l \cap G_{l'} \neq \emptyset$, $l \neq k$. However, we require that the discrete solution be independent of how G is decomposed into rectangles.

Grids at different levels in the grid hierarchy must be "properly nested." This means

- (i) a fine grid starts and ends at the corner of a cell in the next coarser grid.
- (ii) There must be at least one level $l-1$ cell in some level $l-1$ grid separating a grid cell at level l from a cell at level $l-2$, in the north, south, east, and west directions, unless the cell abuts the physical boundary of the domain.

Note that this proper nesting is not as stringent as requiring a fine grid to be contained in only one coarser level grid. For example, in Fig. 2.1, there is one grid at level 3, $G_{3,1}$. Every grid point in $G_{3,1}$ is contained in one of the two grids at level 2, $G_{2,1}$ or $G_{2,2}$.

Grids will be refined in time as well as space, by the same mesh refinement ratio r , where $r = \Delta x_{l-1} / \Delta x_l$. Thus,

$$\frac{\Delta t_l}{\Delta x_l} = \frac{\Delta t}{\Delta x} = \dots = \frac{\Delta t_1}{\Delta x_1}$$

and so the same explicit difference scheme is stable on all grids. This means more

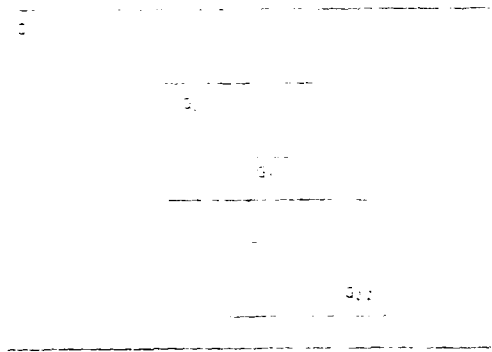


FIG. 2.1 Grid $G_{3,1}$ spans two coarser grids but is properly level nested.

time steps are taken on the finer grids than on the coarser grids. This is a reasonable requirement from the point of view of accuracy, since for many difference schemes, the leading terms in the spatial and temporal truncation error are of the same order. In addition, the smaller time step of the fine grid is not imposed globally. In this implementation we only allow an even refinement ratio. This simplifies the error estimation procedure described later, by avoiding the need of distinguish between an odd and even number of grid points in a grid.

At discrete times the grid hierarchy may be modified. The finest grids need to be changed ("moved," deleted if necessary) most often. When grids at level l are changed, all finer level grids are changed as well, but the coarser grids may remain fixed. New grids at level l may replace the old ones, but they are still subject to the same "proper nesting" requirement.

A point $(x, y) \in D$ may be contained in several grids. The solution $u(x, y)$ will be taken from the finest grid containing the point. If there are several equally fine grids containing the point, any fine grid value will suffice, since the solution on the intersection of overlapping fine grids will be identical.

3. INTEGRATION ALGORITHM

AMR assumes there is a basic, underlying, conservative, explicit finite difference scheme of the form

$$u_{i,j}^{n+1} = u_{i,j}^n - \frac{\Delta t}{\Delta x} (F_{i+1/2,j} - F_{i-1/2,j}) - \frac{\Delta t}{\Delta y} (G_{i,j+1/2} - G_{i,j-1/2}). \quad (1)$$

The values $u_{i,j}$ are cell-centered quantities. Each cell is defined by its four corner grid points. If there are no refined regions, then Eq. (1), augmented by the discretized physical boundary conditions, defines the time evolution on a single grid.

With multiple grids, each grid is separately defined and has its own solution vector, so that a grid can be advanced independently of other grids, except for the determination of its boundary values (see Section 4). The integration steps on different grids are interleaved, so that before advancing a grid to time $t + \Delta t$, all the finer level grids have been integrated to time t . Scheme (1) is still initially applied on every grid at every level, but the results will need to be modified in case

- (i) the cell is overlaid by a finer level grid; or
- (ii) the cell abuts a fine grid interface but is not itself covered by any fine grid.

In case (i), the coarse grid value at level $l-1$ is defined to be the conservative average of the fine grid values at level l that comprise the coarse cell. After every coarse integration step, the coarse grid value is simply replaced by this conservative

average, and the value originally calculated using (1) is discarded. For a refinement ratio of r , we define

$$u_{i,j}^{\text{coarse}} \leftarrow \frac{1}{r^2} \sum_{p=0}^{r-1} \sum_{q=0}^{r-1} u_{k+p, m+q}^{\text{fine}},$$

where the indices refer to the example in Fig. 3.1. We could define a coarse cell u at multiples of the fine time step in the same way, but this is not necessary. This is equivalent (within roundoff error) to redefining the coarse fluxes around the overlayed coarse grid point to be the sum over the fine time steps of all fine grid fluxes calculated on any boundary segment for that cell. However, this implementation would use extra storage to save the fine grid fluxes. By updating the solution values themselves, no extra flux storage is needed.

In case (ii), the difference scheme (1) itself that is applied to the coarse cell must be modified. According to (1), the fine grid abutting the coarse cell has no effect. However, for the difference scheme to be conservative on this grid hierarchy, the fluxes into the fine grid across a coarse-fine cell boundary must equal the flux out of the coarse cell. (This conservative procedure has been discussed by [17]. A fuller discussion of conservation at grid interfaces is in [4].) We use this to redefine the coarse grid flux in case (ii). For example, in Fig. 3.2, the difference scheme at point i, j should be

$$\begin{aligned} u_{i,j}(t + \Delta t_{\text{coarse}}) \\ = u_{i,j}(t) - \frac{\Delta t_{\text{coarse}}}{\Delta x} \left[F_{i+1/2,j}(t) - \frac{1}{r^2} \sum_{q=0}^{r-1} \sum_{p=0}^{r-1} F_{k+1/2, m+p}(t + q \Delta t_{\text{fine}}) \right] \\ - \frac{\Delta t_{\text{coarse}}}{\Delta y} [G_{i,j+1/2}(t) - G_{i,j-1/2}(t)], \end{aligned} \quad (2)$$

where Δx and Δy are coarse spatial step sizes. The double sum is due to the refinement in time: for a refinement ratio r , there are r times as many steps taken on the fine grid as the coarse grid. If the cell to the north of (i, j) were also refined, the flux $G_{i,j+1/2}$ would be replaced by the sum of fine fluxes as well.

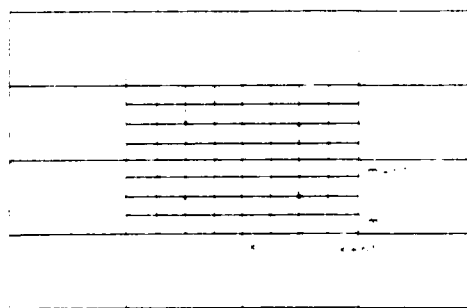


FIG. 3.1. The coarse cell value is replaced by the average of all the fine grid points in that cell

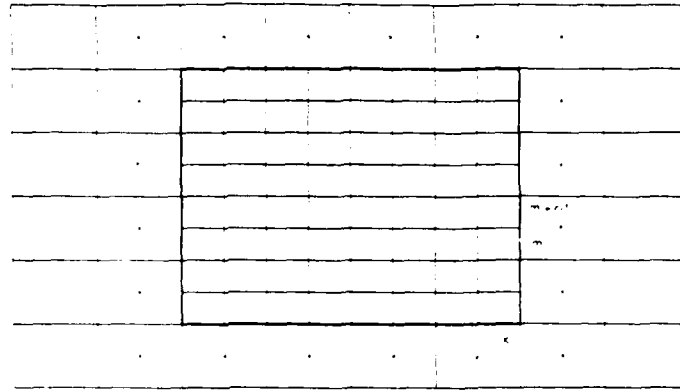


FIG. 3.2. The difference scheme is modified at a coarse cell abutting a fine grid.

This modification is implemented as a correction pass applied after a grid has been integrated using scheme (1) and after the finer level grids have also been integrated, so that the fine grid fluxes in (2) are known. The provisional coarse flux used in (1) is subtracted from the solution $u_{i+1,j}^{\text{coarse}}(t + \Delta t_{\text{coarse}})$, and the fine fluxes are added using Eq. (2). To implement this modification, we save an array δF of fluxes at coarse grid edges corresponding to the outer boundary of each fine grid. After the coarse grid fluxes have been calculated by (1), we initialize δF with

$$\delta F_{i+1/2,j} := -F_{i+1/2,j}^{\text{coarse}}. \quad (3)$$

At the end of each fine grid time step, we add to $\delta F_{i+1/2,j}$ the sum of the fine grid fluxes along the $(i+1/2, j)^{\text{th}}$ edge.

$$\delta F_{i+1/2,j} := \delta F_{i+1/2,j} + \frac{1}{r^2} \sum_{p=0}^{r-1} F_{k+1/2,m+p}^{\text{fine}}.$$

Finally, after r fine grid time steps have been completed, we use $\delta F_{i+1/2,j}$ to correct the coarse grid solution so that the effective flux is that of (2). For example, for cell $(i+1, j)$, we make the correction

$$u_{i+1,j}^{\text{coarse}} := u_{i+1,j}^{\text{coarse}} + \frac{\Delta t_{\text{coarse}}}{\Delta x_{\text{coarse}}} \delta F_{i+1/2,j}.$$

If the cell $i+2, j$ were refined, we would also make the correction

$$u_{i+2,j}^{\text{coarse}} := u_{i+2,j}^{\text{coarse}} - \frac{\Delta t_{\text{coarse}}}{\Delta x_{\text{coarse}}} \delta F_{i+3/2,j},$$

and similarly for the vertical fluxes.

The boundary fluxes δF are stored in a vector associated with every fine grid. In the initialization step (3), there may be several coarse grids that set δF . Since all fluxes calculated at a given edge and level are identical (up to roundoff error) and are independent of the particular grid on which they are calculated, we simply use the last value assigned. At the end of a time step, we may have several fine grids available to update a given coarse cell edge, since overlapping grids are permitted. For this reason, we use a matrix to indicate the edges of a coarse cell that have already been updated and only perform the update once for each edge. As before, it does not matter which fine grid actually performs the update for any given edge, so the result is independent of the order in which the fine grid list is traversed. This modification is a negligible amount of work, taking approximately 0.3% of a typical run time. On machines with a scatter gather operation, this should proceed even faster.

We emphasize that this work is done as a "fix up" step after each grid is updated using scheme (1). In this way, the integrator can be separated from the additional work which is needed because of the grid hierarchy. A new difference scheme can be substituted by a user unfamiliar with and not interested in the inner workings of the AMR program.

4. BOUNDARY CONDITIONS

A discussion of boundary conditions completes the description of the integration procedure on a multiple grid hierarchy. Let the interior integration scheme have a stencil which is centered in space, with d points to each side. To compute the new time step, AMR provides solution values at the old time step on a border of cells of width d intersected with the physical domain. The user must supply the code to compute any additional information needed to implement the boundary conditions. For example, if boundary conditions are imposed by extrapolation, the user would provide the extrapolated values for points outside the domain.

For a grid at level l , the bordering cell values are provided using values from adjacent level l grids where they are available; otherwise, the AMR algorithm computes boundary values using bilinear interpolation from coarser level solution values. If necessary, we also interpolate linearly in time.

It may happen that a point (x, y) is inside the domain D , but one or more surrounding coarse grid points needed for the bilinear interpolation are outside. As before, we assume there is a user-supplied routine that can provide exterior coarse grid points given some interior points.

Our implementation partitions the required border cells at level l into rectangular boundary patches. For each rectangular piece we:

- (i) find solution values from level $l-1$ grids on a slightly larger rectangular piece enclosing the border cells;
- (ii) linearly interpolate for the border values;

(iii) if there are fine grids at level l that could supply some values (say an adjacent fine grid), overwrite the linearly interpolated values from step (ii).

In step (i), most of these coarser level values are found by intersecting the rectangular patch with level $l-1$ grids and by filling the overlapping pieces. However, it may be necessary to go to even coarser grids to supply these level $l-1$ values. This is done by applying (i) to (iii) recursively to the smallest rectangular patch containing the unfilled cells.

For efficiency, it is important that the boundary values are supplied on a rectangular patch at one time and not computed a point at a time. Even though the amount of work is proportional to the boundary of each grid, our initial implementation took 40% of the run time and had to be rewritten. By working on grid patches, the bulk of the memory transfers are done in blocks, and the number of subroutine calls is minimized. This is particularly important on the Cray, where there is a substantial performance penalty for single word accesses and subroutine calls.

5. CREATING THE GRID HIERARCHY

At specified time intervals, an error estimation procedure is invoked, and a new grid structure is determined. If there are several nested levels of refined grids, the error estimation and grid generation procedures are recursively applied on each level, from finest to coarsest, to (re-)create the fine grids at the next level. The error estimation procedure (see Section 6) produces a list of coarse grid points with large error estimates, indicating that a fine grid patch is needed in that region. Every flagged coarse grid point should be included in a finer grid. Our grid generation algorithms try to produce grids that have as little overlap as possible, so that the area that is unnecessarily refined is as small as possible. The algorithm also strives for a small number of patches that are as large as possible, to reduce the computational overhead. It is difficult to find a foolproof algorithm that satisfies these often conflicting goals. However, we have developed heuristics that have been successfully tested in many different applications. A much fuller discussion of grid generation is in [3]. Here, we will describe the particular set of algorithms that produced the numerical results in Section 7.

Suppose there is a base level, l_{base} , where grids will stay fixed, but that the finer levels from $l_{\text{base}} + 1$ to l_{finest} may be "moved." Starting with the finest level grids, we estimate the error, using a procedure described next. If there are points where the error estimate is too high, these points are flagged, and a level $l_{\text{finest}} + 1$ grid will be needed. Next, we estimate the error on the existing $l_{\text{finest}} - 1$ grids. If there are flagged points, a different level l_{finest} grid will be created, making sure that if there are any level $l_{\text{finest}} + 1$ grids, they are properly contained in the level l_{finest} grids. This continues until the error is estimated on the base level grids. Thus, it is only possible to add one new level at a time, although many levels may be removed

during a single regridding operation. (At the initial time however, where the initial data is known for all x, y and not just at coarse grid points, it is possible to add many levels at a time. This is essential for some problems, where the error can depend entirely on the initial conditions.)

In more detail, our regridding algorithm performs the following steps:

(1) *Adds the buffer zone.* A buffer zone of unflagged points is added around every grid. This ensures that discontinuities or other regions of high error do not propagate out from a fine grid into coarser regions before the next regridding time. This is possible because of the finite propagation speed of hyperbolic systems. The larger the buffer zone, the more expensive it is to integrate the solution on the fine grids, but the less often the error needs to be estimated on the coarse grids and the fine grids moved. The buffer zone is added by flagging all coarse grid points that are sufficiently close to flagged points with high error estimates. A buffer zone of two cells in each direction is typical. By flagging neighboring points, instead of enlarging grids at a later step, the area of overlap between grids is reduced.

(2) *Flags every cell at level l corresponding to an interior cell in a level $l+2$ grid.* This will maintain proper nesting, by ensuring that there will be a new level $l+1$ grid containing every point in the level $l+2$ grid, even if the level l grid error estimation did not report a high error. This procedure ensures that the fine grid error estimates are used instead of the coarse grid estimates at the same point. To ensure proper nesting, points within one cell of a non-physical (interior) boundary of G are deleted from the list of flagged points.

(3) *Creates rectangular fine grids.* The grid generator takes all the flagged points as input, and outputs a list of corners of rectangles that are the level $l+1$ grids. Nearby points are clustered together, and a fine grid patch spanning each cluster is formed. These clustering algorithms use heuristic procedures described separately below.

(4) *Ensures proper nesting.* The new fine grids are checked to ensure that they are properly contained in the base level grids. If they are not, the new grid is repeatedly subdivided until each piece does fit. Since the flagged points originally were inside the base grid, at least one cell from the boundary, the new grid containing the flagged points must eventually lie inside as well. Since the base level grids did not move, step (2) cannot be used to ensure the proper nesting of this level. This problem only arises when the base grids are a non-convex union of rectangles.

Step (3) is the difficult one. Since problems in gas dynamics develop 1-dimensional discontinuities, we have streamlined the more general grid generation procedures of [3] for this particular application. The procedure we use here includes a *bisection* step and a *merging* step. Initially, a grid patch is formed around the entire list of flagged cells on a given level. The *efficiency* of the patch is measured by taking the ratio of flagged cells to the total number of cells in the new grid. If this efficiency rating is less than an input minimum efficiency (e.g., 60%), the long direction of the rectangular grid is bisected, and the flagged points are

sorted into two clusters depending on which half they are in. The process is repeated on the two clusters. The bisection step ends when each cluster has an acceptable efficiency rating.

The bisection step uses no geometric information, so although each grid may be "acceptable" by itself, the resulting grid hierarchy may not be optimal. For this reason, the bisection is followed by a merge step. In addition to an absolute efficiency criterion, grids are merged if the new grid is relatively more efficient than the two smaller grids. The cost function we use to measure this is proportional to the cost of an integration step on each grid. On an m by n grid, $(m+1)$ by $(n+1)$ fluxes are calculated, with perhaps 1000 vector operations per flux. In addition there is a cost associated with the perimeter of each grid: finding interface conditions, conservative updating of coarser grids, and special slope calculations that are done only for boundary fluxes. Some of this work uses scalar arithmetic, at least on machines such as the Cray 1 that does not vectorize indirect addressing and gather/scatter operations. The total cost associated with a grid is proportional to $mn + m + n$. Grids are merged if the single resulting grid has a smaller cost. The merging step ends when no pair of grids can be successfully merged.

Although this procedure is somewhat ad hoc, it has been successfully used on several different types of problems. The grid generation routines, not including the solution initialization on each grid or the error estimation to produce the flagged points, account for approximately 1.7% of the CPU time for a typical run.

6. ERROR ESTIMATION

In [6], estimates of the local truncation error were used to select those grid points on a given level with unacceptably large errors. If the solution $u(x, t)$ is smooth enough, the local truncation error $u(x, t+k) - Qu(x, t)$ on a mesh with spatial step h and time step k satisfies

$$\begin{aligned} u(x, t+k) - Qu(x, t) &= k(c_1(x, t)k^q + c_2(x, t)h^q) + kO(k^{q+1} + h^{q+1}) \\ &\equiv \tau(x, t) + kO(k^{q+1} + h^{q+1}), \end{aligned}$$

where the leading term is denoted by τ . Here we assume our difference method Q has order of accuracy q in both time and space. If u is smooth enough, then if we take two time steps with the method Q , to leading order the error is 2τ .

$$u(x, t+2k) - Q^2u(x, t) = 2\tau + kO(k^{q+1} + h^{q+1}).$$

Let Q_{2h} denote the same difference method as Q but based on a mesh widths of $2h$ and $2k$. Then

$$u(x, t+2k) - Q_{2h}u(x, t) = 2^{q+1}\tau + O(h^{q+2}).$$

LOCAL ADAPTIVE MESH REFINEMENT

By taking two steps with the regular integration scheme, and one "giant" step using every other grid point, the difference

$$\frac{Q^2 u(x, t) - Q_{2h} u(x, t)}{2^{q+1} - 2} = \tau + O(h^{q+2})$$

gives an estimate of the local truncation error at time t . We emphasize that it is not necessary to know the exact form of the truncation error (e.g., h^{q+2}), only its order.

This procedure is easily implemented in AMR for the conservative finite difference methods presented here. The values on a grid at a given level are projected onto a virtual grid coarsened by a factor of two in each direction. The solution on both grids is advanced in time: the original grid for two time steps, the coarsened grid for one step using a time step twice as large. The difference between the solutions obtained on the two grids at each point is proportional to the local truncation error at that point. At coarse cells where the difference between the two sets of values exceeds some tolerance, all four cells contained in the real grid are flagged as requiring refinement. Notice that this estimation procedure is independent of the finite difference method actually used, as well as the pde. One disadvantage of this procedure is that it always predicts a large error in the neighborhood of captured discontinuities. It is easy to construct examples for which the procedure outlined above will give values on the coarsened grid which differ pointwise by an amount independent of the mesh spacing in the neighborhood of a shock. In general, this leads to refinement of the mesh at all discontinuities with strength greater than some minimum.

Theoretically, one could define a distributional error by averaging the difference between the two solutions over some region centered at the given cell whose size is $O(1)$ relative to the mesh spacing. We have devised various techniques for carrying out such a procedure, all of which are equivalent to ignoring the pointwise error estimate in the neighborhood of those discontinuities which, by some other criteria, are considered adequately resolved. For problems in shock hydrodynamics, only shock discontinuities, and not slip surfaces or contact discontinuities, are expected to satisfy any such criteria. This is because conservative finite difference methods applied to shocks mimic the convergence of characteristics in the analytic solution, so that the shock spreads only over a fixed number of zones independent of the mesh spacing and time. Thus, for example, it is unnecessary to refine the grid in the neighborhood of a shock separating two constant states. In contrast, it is necessary to refine at linearly degenerate discontinuities since the number of cells over which they spread is an increasing function of time.

There is a second set of difficulties with refinement in the presence of strong shocks. There is evidence that shock-capturing methods are zeroth-order methods, i.e., that the fluxes computed in the neighborhood of the shock differ by $O(1)$ from the exact fluxes at a given time step [21, 16]. These $O(1)$ errors could generate

waves associated with the characteristic families crossing the shock, sending $O(1)$ pointwise errors into the postshock region. For shocks computed on a single uniform grid, this does not occur, because the same $O(1)$ errors are committed on successive cell edges with a phase lag, so that errors in the time integral of successive fluxes cancel upon differencing. However, when a shock intersects an interface between two grids with different mesh spacings, the $O(1)$ errors in the time integrals of the fluxes on each of the two grids will be different, generating $O(1)$ errors in the solution propagating into the postshock state. In practice, we have observed that the amplitude of the spurious waves generated in this fashion is proportional to the amplitude of the jump in the characteristic quantities carried by characteristics crossing the shock. In practice, then, there is usually a threshold shock strength below which the errors generated by a shock crossing a grid discontinuity are acceptable and above which the errors generated are too large. For the latter shocks, they must be refined everywhere, if they are to be refined anywhere.

We illustrate this with an example where we force the algorithm not to refine the grid above a certain height. This forces the strong incident shock, with a shock Mach number of 10, to pass through a fine grid boundary into the coarse grid. The oscillations caused by this are apparent in the contour plots of Fig. 6.1 and the plot of Fig. 6.2 for a fixed value of y .

Combining the two considerations given above, a fairly general refinement strategy is to use the local truncation error estimate described above, but ignore it in the neighborhood of gas-dynamic shocks whose strength lies below some predetermined threshold. This has the effect of refining, possibly unnecessarily, all shocks whose strength is above the threshold. We have found that this strategy works acceptably well if the coarsened base grid is sufficiently fine, so that the waves are well separated. A much simpler strategy, which is applicable in a large class of problems, is simply to use the user's knowledge of the problem instead of the truncation error estimates. For example, in the shock reflection problems given below, the solution is made up entirely of smooth waves and weak shocks a certain distance behind the incident shock. Consequently, we simply ignore the local truncation error estimate in that region.

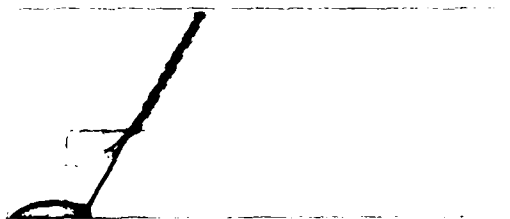


FIG. 6.1 Contour plot showing the effects of a strong shock passing through a grid boundary.

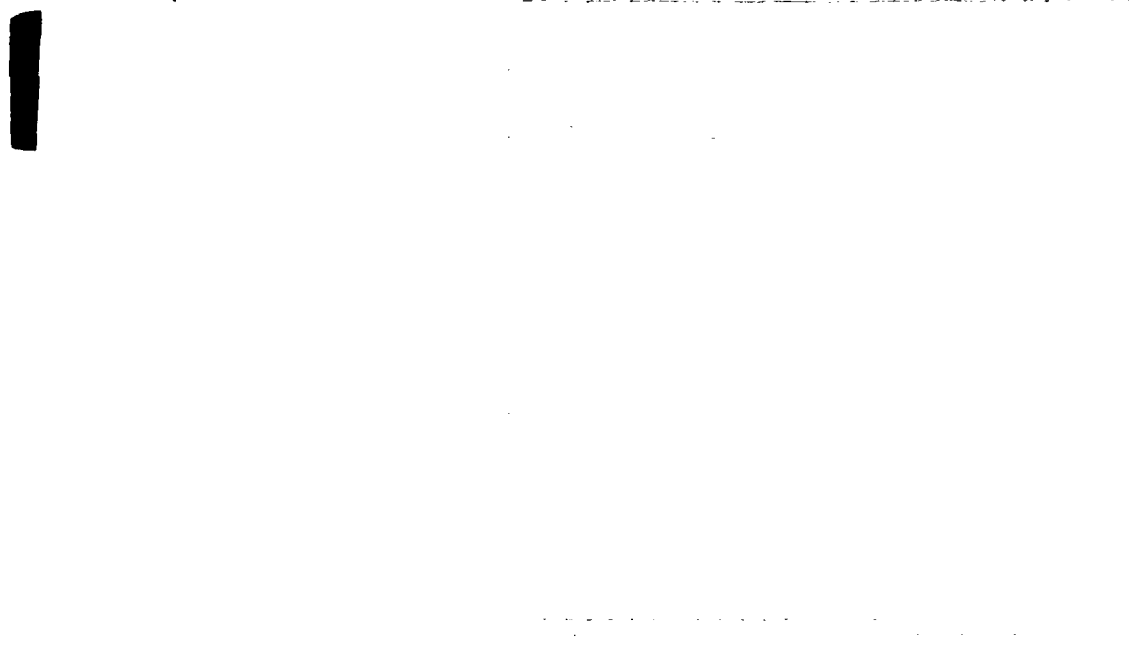


FIG. 6.2 Density profile for a horizontal line slightly below the grid interface intersecting the shock

7. NUMERICAL RESULTS

We choose as our test problem reflection of a planar shock in an ideal gas by an oblique surface. In this problem, a straight shock is incident on a perfectly reflecting surface. At later times, a reflected wave pattern is generated, depending only on α , M_1 , and γ , where α is the angle between the direction of propagation and the reflecting surface, M_1 is the incident shock Mach number, and γ is the ratio of specific heats. The solution to this problem is formally self-similar, depending on (x, t) only in the combination $(x/t, y/t)$. Thus, the time dependence of the solution is given by a linear scaling of a fixed wave pattern with time. We are particularly interested in values of the problem parameters for which very complicated small scale structures are observed.

The underlying integration method in our AMR calculations is a second-order Godunov method described in [10]. In our calculations, we make use of the fact that the regions where the small scale behavior can appear are localized in the vicinity of the reflection point. Our procedure for estimating the error is to measure the local truncation error of the density, except that we do not tag points beyond a certain distance behind the incident shock. This effectively restricts our refinement

TABLE I

Breakdown of Computational Times Obtained Using FLOWTRACE

Grid integration	78.29
Interpolation	12.95
Output	2.87
Grid updates	2.78
Grid generation	1.71
Memory management	0.59

to be in a window moving with the reflection point, and shuts off the grid refinement around the (weak) reflected shock.

The results presented here were calculated on the Cray XMP 22 at the LLNL NMFECC, using the CFT 1.14 compiler. To obtain detailed diagnostic information about where most of the time is spent in the calculation, we used the FLOWTRACE option of the CFT compiler in the first calculation below. The total time spent was 5674 seconds of CPU time. Table I shows a breakdown of the calculation time into six categories: the integration routine, the interpolation routines (for constructing boundary conditions and initializing new fine grids), the updating routines (fine grids updating coarse grids and for maintaining conservation across grid interfaces), the grid generation routines, output routines, and memory management routines.

The main result is that the integration step takes about 80% of the computational time. This figure includes integration steps needed for the error estimation. However, measurements show that the latter is only 3% of the integration cost, with actual useful integration steps accounting for 97% of the integration time. Note that the error estimation cost is very small despite the fact the error is estimated at every other coarse time step. There are two reasons for this. First, over 90% of the cells being integrated belong to the finest level grids (level 3 in both calculations), and the error is not estimated there. Second, since refinement is performed in time as well as space, the overwhelming majority of the work is done on the finest grid. Table II shows the number of cell updates done on each grid level, as well as the total number of cell updates done for error estimation.

We can thus obtain a rough estimate of the efficiency of AMR relative to computing on a uniform grid. About 80% of the run time is spent integrating grids. The

TABLE II

Number of Cell Updates at Each Level

Level 1	2.98×10^5
Level 2	4.59×10^6
Level 3	1.13×10^8
Error estimation	3.06×10^6

finest level grids occupy only about 10% of the domain. Thus an equivalent uniform grid computation would require a factor of 8 more CPU time. Of course in this particular problem, one could omit computations ahead of the incident shock, since the solution there is constant, saving approximately half the uniform grid time. In general, this could not be done.

It is more difficult to compare memory usage with that of a uniform grid calculation. The integration algorithm used here requires five 2-dimensional grid arrays for scratch space, in addition to the four required to store the conserved quantities, so that the memory requirements for a uniform grid calculation would be $9 \cdot 320 \cdot 1600 \approx 4.5 \times 10^6$ words. In contrast, the maximum storage used in the calculation performed here was 8.94×10^5 words. Much of the memory use in AMR is due to saving two time levels of the solution on each grid. It is possible to avoid the memory overhead of having full grid scratch arrays by breaking the calculation into pieces. (In fact, we effectively do this in the AMR calculations by restricting the size of any grid to be less than some pre-determined maximum, subdividing grids that are too large.) However, this would introduce overheads and programming complexities in the uniform grid calculation similar to those in AMR. In any case, even if those overheads could be neglected and only four full grid arrays were required, the memory required would be 2.0×10^6 words, a factor of 2.2 larger than that required by AMR.

In Fig. 7.1, we show results for the case $M_\infty = 10$, $\alpha = 30^\circ$, $\gamma = 1.4$. The domain is a rectangle of length 2.0 by 0.4, with initial coarse grid spacing $\Delta x = \Delta y = 0.02$. The calculation ran for 149 coarse grid time steps. The error was estimated every other step, with a buffer zone of one cell and a grid efficiency of 65%. The error tolerance was 0.02. The mesh was refined by a factor of 4 in each direction at each grid level. The finest grids in this calculation represent a factor of 4 increase in resolution in each spatial direction over the finest grid calculation in [22]. Figure 7.1a shows the location of the level 2 and 3 grids at time $t = 1.20$. In displaying the solution, we show two sets of plots. One is a contour plot of the full flow field. The other is an enlargement of the region around the reflection point. This is the part of the domain covered by the level two and three grids. In both cases, the contour plots are made using the finest available grid in the subregion. Due to the increased resolution, we can now observe a non-self-similar Kelvin-Helmholtz rollup along the principal slip line. This is to be expected, since this slip line is unstable. The Kelvin-Helmholtz rolls are formed near where the weak shock emanating from the second triple point impinges on the slip line. They then propagate along the slip line and are eventually swept up into a large rollup at the tip of the jet, along the bottom wall.

Finally, in Fig. 7.2 we present results for $M_\infty = 8$, $\alpha = 35^\circ$, $\gamma = 1.107$. It has been noticed [11] that the wave patterns associated with double Mach reflection become increasingly complex as γ is reduced. The jet along the reflecting wall formed by the slip line from the principal Mach triple point is more and more strongly accelerated, pushing the Mach stem out ahead of it. This leads to strongly rotational supersonic flow and the formation of multiple Mach triple points. The pre-

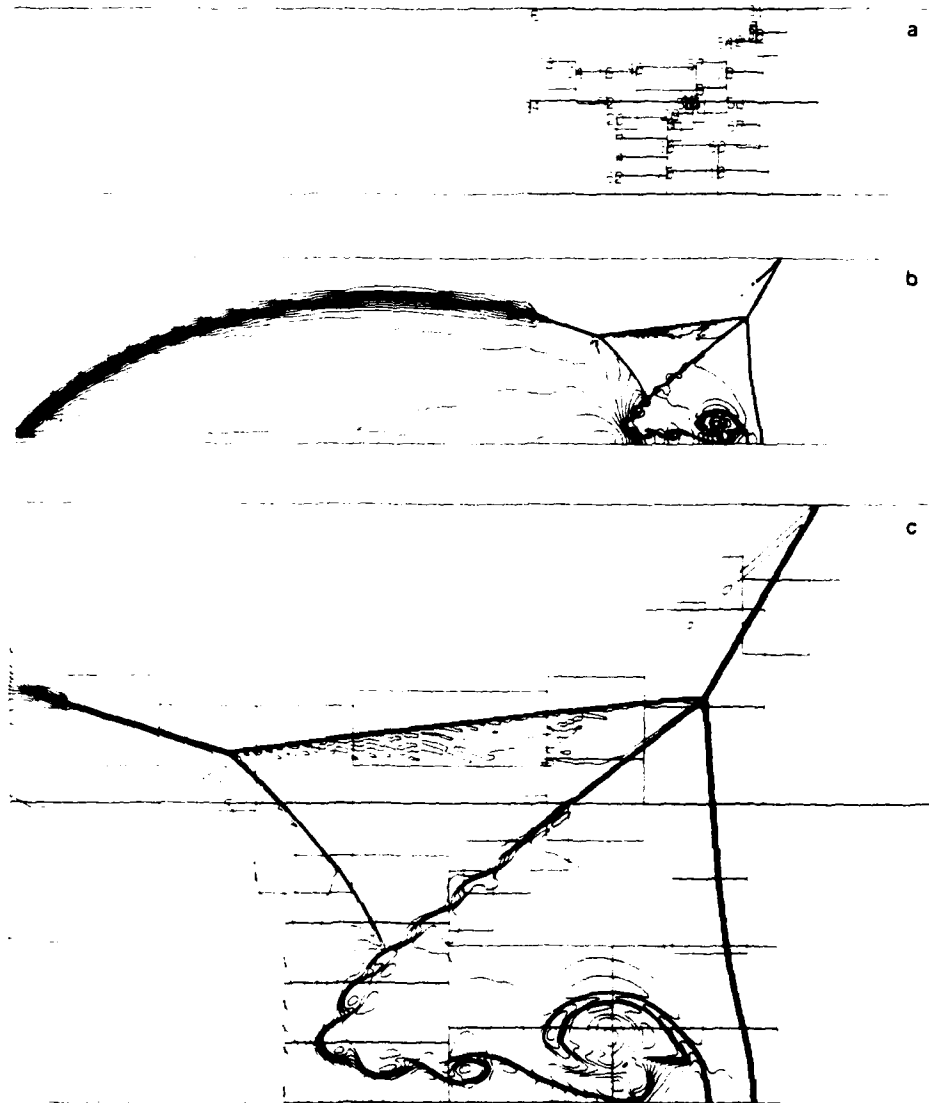


FIG. 7.1. Shock reflection off an oblique wedge with $\gamma = 1.4$, $M_\infty = 10$, $\alpha = 30^\circ$: (a) Shows the grid hierarchy: grid 1 is a level 1 grid, grids 6 and 34 are level 2 grids, and the rest are level 3 grids, at time $t = 0.12$. (b) Density, full flow field. (c) Density, level 2 and 3 grids only. (d) Pressure, level 2 and 3 grids only. (e) Entropy, level 2 and 3 grids only.

LOCAL ADAPTIVE MESH REFINEMENT

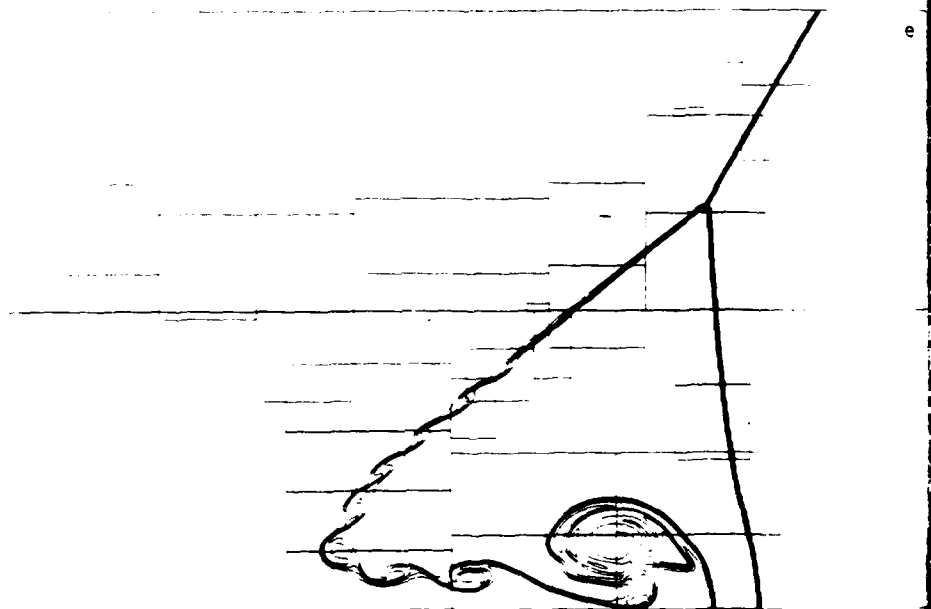
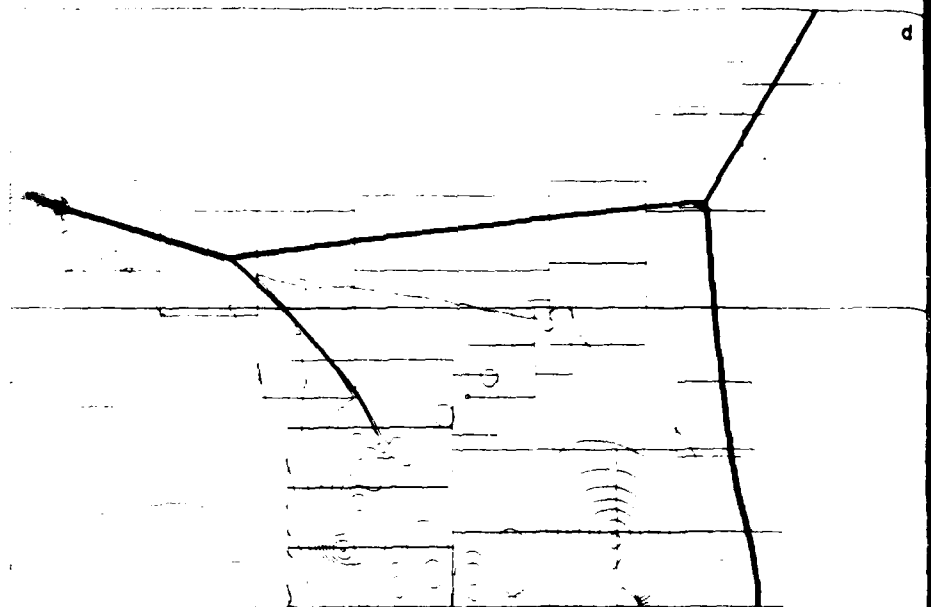


Fig. 7.1—Continued

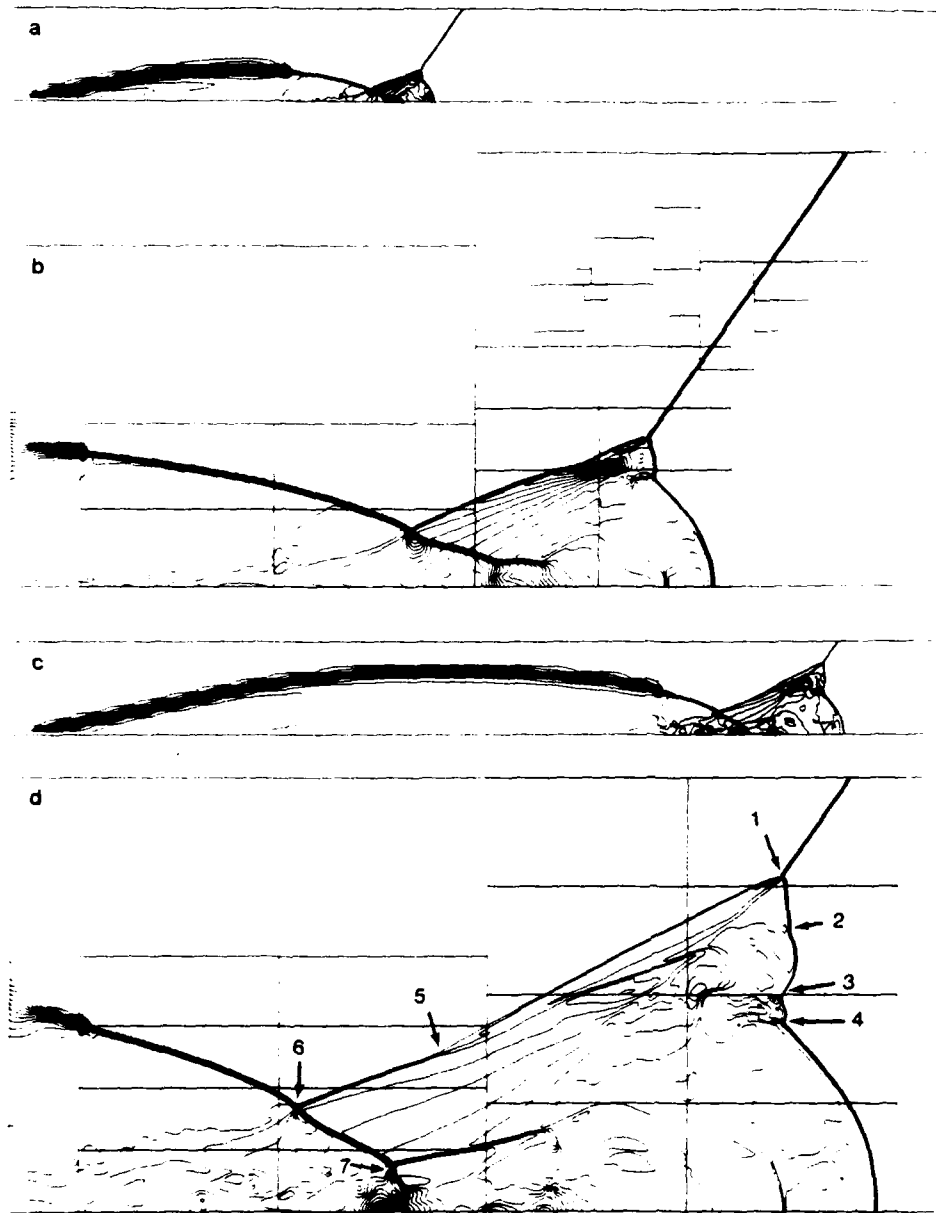


FIG. 7.2. Shock reflection off an oblique wedge with $\gamma = 1.107$, $M_1 = 8$, $\alpha = 35^\circ$: (a) Density, full flow field, at $t = 0.115$. (b) Density, level 2 and 3 grids only, $t = 0.115$. (c) Density, full flow field, at $t = 0.230$. (d) Density, level 2 and 3 grids only, $t = 0.230$.

sent results represent a rather extreme example of this, with a total of seven Mach triple points in the double Mach region, including a third triple point along the top shock. The seven triple points are marked in Fig. 7.2d. This calculation gives some indication of why adaptive mesh refinement is an important tool in these problems. As γ approaches one, the distance between the leading edge of the wall jet and the main Mach stem becomes smaller and smaller, requiring more grid resolution in that region. The value of γ in this calculation represents the limit of resolution, given the resources available on the Cray XMP. The calculations in [11] using uniform grids without mesh refinement, were not fully resolved for $\gamma < 1.25$. Even with mesh refinement, at this low value of γ the flow field is not fully resolved at time $t = 0.115$, in Fig. 7.2a and b. Only after running to time $t = 0.230$, which by self-similarity corresponds to increased grid resolution, is the solution adequately resolved.

8. CONCLUSIONS

The complexity of our AMR code might be intimidating to a new user. Not counting the integration routine, our program consists of 3000 lines of Fortran. However, a big code is not necessarily a fragile code. We have been careful to develop AMR to make it automatic and robust. In addition, a user should be able to use AMR without having to understand it all. This makes it important to develop AMR in a modular way. A user should be able to plug in an integrator for a new problem without knowing details about how the more computer science oriented parts of AMR work, but knowing that these other parts will indeed work. We have already demonstrated this modularity by using AMR to compute transonic flow in conjunction with FLO52 [5] and to compute a combustion problem with a simple induction time model for chemistry [2].

The most difficult problems will best be solved by combining several adaptive techniques. Despite its more complicated data structures, AMR has already been combined with the conservative front-tracking scheme of [9]. This enables tracking of a strong incident shock, while using shock-capturing for the other discontinuities, and avoids the mismatch of strong captured shocks crossing grid boundaries. This combined approach is being used to study transition from regular to Mach reflection. Finally, we intend to couple this method with the variational technique of [8]. Their mesh-moving technique would allow the underlying mesh geometry to be approximately aligned with global features in the flow, leading to more efficient refined meshes. However, the actual mesh refinement for error reduction would be done with AMR, so the global time step penalty of moving mesh methods is not incurred. Lastly, a major open question is how to use implicit difference schemes with embedded grids for a time-dependent calculation. This will be needed to compute solutions to hyperbolic-parabolic problems, such as the Navier-Stokes equations at high Reynolds number.

ACKNOWLEDGMENTS

We thank Michael Welcome for assembling the graphics program for multiple grids used to display the numerical results. We thank John Bolstad for a careful reading of the manuscript. The first author's work was supported in part by the Department of Energy Contract DEAC0276ER03077-V and by the Air Force Office of Scientific Research under Contract AFORSR-86-0148. The second author's work was supported in part by the Office of Energy Research of the U.S. Department of Energy at the Lawrence Livermore National Laboratory under Contract W-7405-ENG-48 and at the Lawrence Berkeley Laboratory under Contract DE-AC03-76SF00098 and by the Air Force Office of Scientific Research under Contract AFOSR-ISSA-870016. Phillip Colella wishes to thank the Courant Institute, which he visited for five months under Department of Energy Contract DEAC0276ER03077-V.

REFERENCES

1. S. ADJERID AND J. FLAHERTY, RPI Computer Science Report No. 85-21 (unpublished).
2. J. BELL, P. COLELLA, J. TRANGENSTEIN, AND M. WELCOME, presented at the 8th AIAA CFD Conference, June 1987, Honolulu, HI; Lawrence Livermore Report UCRL-96443 (unpublished).
3. M. J. BERGER, *SIAM J. Sci. Statist. Comput.* **7**, 904 (1986).
4. M. J. BERGER, *SIAM J. Num. Anal.* **24**, 967 (1987).
5. M. J. BERGER AND A. JAMESON, *AIAA J.* **23**, 561 (1985).
6. M. J. BERGER AND J. OLIGER, *J. Comput. Phys.* **53**, 484 (1984).
7. J. BOLSTAD, Ph. D. thesis, Department of Computer Science, Stanford University, California, 1982 (unpublished).
8. J. U. BRACKBILL AND J. S. SALTZMAN, *J. Comput. Phys.* **46**, 342 (1982).
9. I. CHERN AND P. COLELLA, *J. Comput. Phys.*
10. P. COLELLA, Lawrence Berkely Laboratory Report LBL-17023; *J. Comput. Phys.*
11. P. COLLELA AND H. GLAZ, in *Proceedings, 9th Intl. Conf. Numerical Methods in Fluid Dynamics*, June 1984; Lecture Notes in Physics Vol. 218 (Springer-Verlag, New York Berlin, 1985).
12. P. COLLELA AND P. WOODWARD, *J. Comput. Phys.* **59**, 264 (1985).
13. H. A. DWYER, AIAA Paper No. 83-0449 (unpublished).
14. W. D. GROPP, *SIAM J. Sci. Statist. Comput.* **4**, 191 (1980).
15. K. MILLER AND R. N. MILLER, *SIAM J. Num. Anal.* **18**, 1033 (1981).
16. W. NOH, Lawrence Livermore National Laboratory Report No. UCRL-52112, June 1976 (unpublished).
17. S. OSHER AND R. SANDERS, *Math. Comput.* **41**, 321 (1983).
18. M. M. RAI AND D. A. ANDERSON, *J. Comput. Phys.* **43**, 327 (1981).
19. W. J. USAB AND E. M. MURMAN, AIAA Paper No. 83-1946-CP, July 1983 (unpublished).
20. K. H. WINKLER, Ph. D. thesis, University of Göttingen, 1976 (unpublished).
21. P. WOODWARD, in *Proceedings Nato Workshop in Astrophysical Radiation Hydrodynamics, Munich, W. Germany, Nov. 1983*, Lawrence Livermore Report UCRL-90009, August 1982 (unpublished).
22. P. WOODWARD AND P. COLELLA, *J. Comput. Phys.* **54**, 115 (1984).



AIAA-89-1930

**An Adaptive Cartesian Mesh Algorithm for the
Euler Equations in Arbitrary Geometries**

M. Berger

Courant Institute of Mathematical Sciences

New York, NY

R. LeVeque

University of Washington

Seattle, WA

**AIAA 9th Computational Fluid
Dynamics Conference**

Buffalo, New York / June 13-15, 1989

An Adaptive Cartesian Mesh Algorithm for the Euler Equations in Arbitrary Geometries

Marsha J. Berger

Courant Institute of Mathematical Sciences
New York University
251 Mercer St.
New York, NY 10012

Randall J. LeVeque

Mathematics Department
University of Washington
Seattle, WA 98195

Abstract

We present a Cartesian mesh algorithm with adaptive refinement to compute flows around arbitrary geometries. Cartesian meshes have been less popular than unstructured or body-fitted meshes because of several technical difficulties. We present an approach that resolves many of these problems. Cartesian meshes have the advantage of allowing the use of high resolution methods that are difficult to develop on unstructured grids. They also allow for efficient implementation on vector computers without using gather-scatter operations except at boundary cells. Some preliminary computational results using lower order boundary conditions are presented.

1. Introduction

The construction of logically rectangular body-fitted grids for complicated geometries is notoriously difficult. One alternative is to use an unstructured mesh, so that the cell volumes are not derived by a smooth mapping from a rectangular domain, as in [16] for example. Another possibility is to simply use a Cartesian grid over the entire flowfield. This introduces the difficulty of imposing solid wall boundary conditions on a grid that is not aligned with the body.

Nonetheless, there are several reasons to prefer the Cartesian grid approach, in addition to the ease of grid generation. First, it allows the use of higher order accurate shock capturing methods that are difficult to achieve on an unstructured mesh with no coordinate directions. A Cartesian grid integrator is highly vectorizable. Gather-scatter type vector operations need to be performed only in a lower dimensional region, and not over the entire flowfield. The basic solver on a Cartesian mesh is also simpler than on a body-fitted mesh, since there are no metric terms. Finally, there is some evi-

dence [20] that for strong shock calculations an unstructured mesh has larger phase errors, and thus poorer shock-capturing abilities, than structured grids.

There are also several difficulties in using Cartesian grids. The main difficulty is the small cell problem. Arbitrarily small cells arise at the edge of the domain where the grid intersects a body. Stable, accurate and conservative difference schemes are needed for these cells. We would like the time step for a time-accurate computation to be based on the cell volume of the regular cells away from the body, and not be restricted by small cells at the boundary. The time step appropriate for the regular grid cells can give a Courant number that is orders of magnitude larger than 1 for the smaller, irregular cells. This will lead to stability problems with standard explicit methods. In this work, we use an approach based on wave propagation that essentially increases the size of the stencil near these small cells and maintains stability for arbitrary time steps. This large time step approach was studied in one space dimension in [13], and has been applied in the present context of small boundary cells in [14,15].

Another problem with Cartesian grids is one of accuracy. Grid stretching, used in body-fitted grids to cluster the grid points in regions where they are needed, cannot be used. Moreover, since the grid is not aligned with the boundary, a loss of resolution may occur near the boundary. To improve the accuracy we use an adaptive mesh refinement algorithm developed in [8]. Rectangular refined grids are superimposed on the coarse grid, so that the efficiency of the integrator on each grid is maintained. The time step is refined along with the mesh width on the fine grids, so that the CFL condition is maintained while allowing larger time steps on the coarser grids. This further concentrates the computational work where it is needed.

Cartesian mesh methods have received increased attention recently. Cartesian grids were used in [19] to solve the full potential equations. This was extended to the Euler equations in two space dimensions in [11], and to three dimensions in [12]. Cartesian grids have also been used in conjunction with an implicit, flux-vector split method for the Euler equations [10]. These calculations, however, suffer from the lack of resolution of a Cartesian mesh; none of the calculations used a local mesh refinement algorithm. The use of a global, tensor product grid to concentrate grid lines near the leading edge of an airfoil, for example, can be very wasteful. In those computations, the small, irregular cells near the body were merged into their neighboring cells to create a cell that was large enough to satisfy a stability constraint. This procedure loses resolution.

In the next section, we describe our overall computational method, including the organization of the calculation and the boundary representation. Section 3 describes the large time step method and the implementation of the solid wall boundary conditions. Section 4 describes the mesh refinement algorithm. In Section 5 we show applications of our method to the inviscid Euler equations in two geometries: shocked flow around two cylinders, and a curved channel calculation.

2. Algorithm and Data Structures

We consider the Euler equations in two space dimensions,

$$U_t + F(U)_x + G(U)_y = 0,$$

where

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}, \quad F(U) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uE + up \end{bmatrix}, \quad G(U) = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vE + vp \end{bmatrix}.$$

and

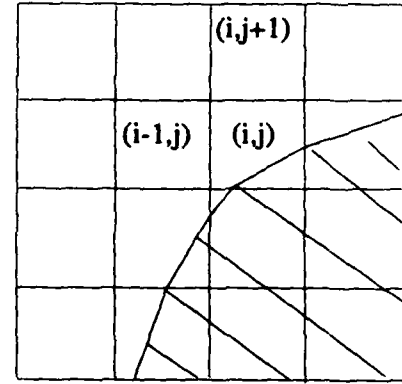
$$p = (\gamma - 1)(\rho E - \rho(u^2 + v^2)/2).$$

The boundary of a solid object is approximated by piecewise linear segments as shown in Figure 1a. We assume that the boundary cuts through each cell at most once, so that the resulting irregular cell is a polygon with at most five sides. We use a finite volume method, with $U_{i,j}$ representing the cell average of the vector of conserved quantities in cell (i,j) , i.e.

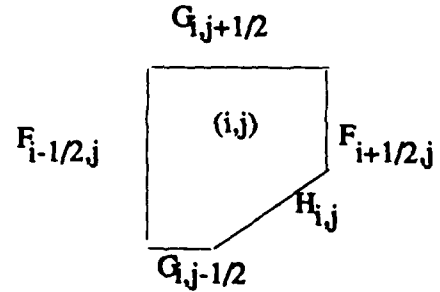
$$U_{i,j}^n = \frac{1}{A_{i,j}} \int_{cell(i,j)} u(x,y,t_n) dx dy,$$

where $A_{i,j}$ is the area of the cell. For regular cells, away from the boundary, $A_{i,j} = \Delta x \Delta y$, but $A_{i,j}$ may be orders of magnitude smaller for cells on the boundary.

All grid cells are indexed using the rectangular Cartesian structure. Additional information about the irregular cells is kept in a linked list data structure that is easily traversed in implementing the boundary conditions. Necessary information includes the cell area and list of vertices for each irregular cell, as well as a pointer to its location in the Cartesian grid. In the other direction, a two dimensional integer array indicates whether a Cartesian cell is regular, and for irregular cells contains a pointer to the corresponding location in the linked list. When grid refinement is used there may be several rectangular grids, each with its own solution storage and irregular points list.



(a)



(b)

Figure 1 (a) a Cartesian grid. The shaded region represents a solid body. (b) Blowup of cell (i,j) showing the fluxes.

In each time step, the cell values are updated by differencing fluxes at the cell sides, as illustrated in Figure 1b. The updating formula is

$$U_{i,j}^{n+1} = U_{i,j}^n - \frac{\Delta t}{A_{i,j}} [F_{i+1/2,j} - F_{i-1/2,j} + G_{i,j+1/2} - G_{i,j-1/2} + H_{i,j}]. \quad (1)$$

boundary of the cell. For example, with Godunov's

method we would obtain $F_{i+1/2,j}$ by solving a Riemann problem $u_t + f(u)_x = 0$ with left and right states $U_{i,j}$ and $U_{i+1,j}$ to determine the correct intermediate state u^* . We then set $F_{i+1/2,j} = h_{i+1/2,j} f(u^*)$, where $h_{i+1/2,j}$ is the length of the interface between cells (i,j) and $(i+1,j)$. For regular cells this length is just Δy . Similarly, $G_{i,j+1/2}$ is the flux per unit time through the top of the cell. Finally, $H_{i,j}$ is the flux per unit time through the irregular side of the cell, which represents the solid wall boundary of the fluid domain. In regular cells, $H_{i,j} = 0$. With higher order Godunov schemes such as MUSCL, the left and right states are modified using slope information to achieve second order accuracy.

In irregular cells there are two difficulties with this approach. First, the neighboring cell values needed to define appropriate slopes may not be present. This is currently handled by setting the slopes to zero, so that the flux reduces to the Godunov flux at these interfaces. Improvements to this algorithm are currently under study.

Even with first order fluxes, there is still a stability problem. Use of these fluxes in updating formula (1) will give instabilities in cells where $A_{i,j}$ is very small relative to Δt . A wave propagation interpretation of this is given in Section 3, where we present a way to modify the fluxes to account for the reflection of waves at the boundary and obtain a much more stable algorithm. First we present an outline of the overall algorithm.

Step 1: Initial flux computation. In the first pass, fluxes at all cell boundaries are computed assuming that the grid is regular, even at interfaces where both neighboring cells lie outside of the actual fluid domain. This is done for ease of vectorization, but the calculations outside the domain will have no influence on the final solution. Also, the interface lengths are always assumed to be Δx or Δy in computing the flux per unit time, regardless of the true length of the side. This will be corrected in step 2 as required.

Step 2: Flux Modification Near the Boundary. In the second step, we march around the solid boundary of the fluid domain, modifying the fluxes of the irregular cells. First we adjust the fluxes F and G at each interface to incorporate the correct length rather than the standard length Δx or Δy . Next, we modify the fluxes to improve the stability of the small cells and incorporate the solid wall boundary conditions. This will be described in more detail in the next section. Finally, we calculate the fluxes $H_{i,j}$ at the irregular side of each boundary cell. This is also described in section 3.

Step 3: Updating $U_{i,j}$. The grid values $U_{i,j}$ are now updated using the flux differencing formula (1).

Step 4: Smoothing at the Boundary. Although the flux modification of Step 2 is intended to give stability

for arbitrarily small cells, in practice very small cells still cause difficulties in some computations. The exact causes of this are currently under study. In the present code, stability is restored by means of an averaging process. In very small cells (those with area less than 3% of the regular cell size, typically), the value of $U_{i,j}^{n+1}$ is replaced by a weighted average of the original value and the value in one or more neighboring cells. The choice of cells depends on the local geometry. The value in the neighboring cell(s) is also modified in such a way that conservation is maintained. The weights used are proportional to the cell areas and hence the value in the small cell is replaced by a value that is essentially equal to that of the cell's primary neighbor (or a weighted averaged of two or three neighboring cells). This procedure has been found to eliminate any remaining instabilities. This algorithm is similar to the flux redistribution algorithm in [9].

3. Boundary Conditions and Flux Modification

For irregular cells at the boundary, the fluxes that are calculated in the first stage of the algorithm are simply the Godunov fluxes obtained by solving the Riemann problem between this cell and each neighbor. In order to understand how these fluxes should be modified for small cells, we first consider Godunov's method on a regular grid cell. The method can be interpreted in the following way: Solve the Riemann problem at each interface to obtain waves propagating away from the interface. For each wave that propagates into the cell, let ΔU represent the jump in the conserved quantities across the wave. Suppose that in time Δt this wave sweeps through a certain fraction α of the cell. Then the cell average $U_{i,j}$ is updated by the quantity $\alpha \Delta U$. Note that the CFL condition requires $\alpha \leq 1$, and that α is the ratio of the area swept out by the wave to the total cell area (see Figure 2). The wave shown in Figure 2 propagates with speed $s > 0$ from the left side of the cell, and so

$$\alpha = \frac{s \Delta t \Delta y}{A_{i,j}} = \frac{s \Delta t}{\Delta x}.$$

Now consider the same wave but suppose that the cell in question is an irregular cell as shown in Figure 3a. We now have $\alpha = s \Delta t / A_{i,j} > 1$, and updating $U_{i,j}$ by $\alpha \Delta U$ would lead to instability. However, an alternative approach that is physically more reasonable is to update $U_{i,j}$ by only $1.0 \Delta U$, since the wave overlaps the entire cell, and then to update cells further to the right by the remainder of the wave $(\alpha - 1) \Delta U$. This is the basis of the large time step method originally described in [13]. In the present context however, there are no cells to the right. Instead, there is a solid wall boundary, off which the wave should reflect. This is illustrated in Figure 3b. The portion of the wave that lies outside the domain is

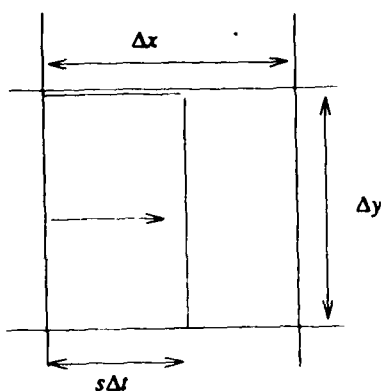


Figure 2 The wave containing the jump in conserved quantities travels to the right a distance $s\Delta t$ away from the interface.

reflected normal to the boundary segment of this cell. Linearization of the solid wall boundary conditions suggests that the reflected wave should carry a jump $\Delta\tilde{U}$, which has the same jump as ΔU in density, pressure and tangential velocity, but which has the jump in normal velocity negated. (Normal and tangential refer to the orientation of the solid wall boundary).

This reflected wave overlaps some fraction $\beta \leq 1$ of the cell, and so $U_{i,j}$ is further updated by $\beta\Delta\tilde{U}$. In addition, the reflected wave may overlap neighboring cells, and each of these is also updated by the fraction of the cell overlapped multiplied by $\Delta\tilde{U}$. In Figure 3b, three neighboring cells are affected by the reflected wave. This is the maximum number possible, so the amount of computational work required is bounded.

As just described, the waves are used to update cell values directly. In the actual implementation, the waves are used to update the fluxes at the cell interfaces by calculating the flux through each interface due to the wave. This makes these boundary conditions easier to use in conjunction with an arbitrary flux differencing method away from the boundaries. The flux at each interface is modified by any wave that crosses the interface. For example, the reflected wave in Figure 3b crosses four interfaces and would modify the flux at each of these interfaces.

Finally, we must calculate the flux $H_{i,j}$ at the solid wall boundary itself. The basic flux is computed by solving a Riemann problem at the wall with data given by $U_{i,j}$ and $\tilde{U}_{i,j}$. The vector $\tilde{U}_{i,j}$ agrees with $U_{i,j}$ in density, pressure and tangential velocity but again has the normal velocity negated. This flux is then modified by any wave that reflects off the wall, once by the outgoing flux of the wave ΔU and then again by the incoming flux of the reflected wave.

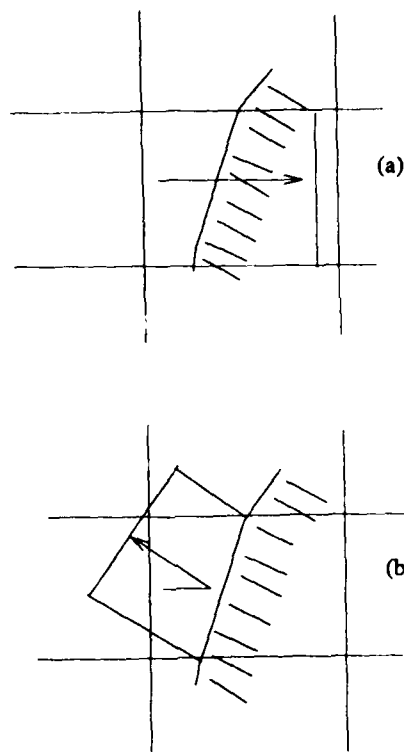


Figure 3 (a) The wave completely sweeps through the small boundary cell. (b) The wave reflects off the boundary back into the domain.

4. Mesh Refinement

The adaptive mesh refinement algorithm (henceforth AMR) is based on the use of uniform, local grid refinements superimposed on an underlying coarse grid. These embedded grid refinements can be recursively nested to maintain a fixed level of accuracy in the calculation. Unlike other embedded grid refinement methods, (e.g. [18]), in this method the grid cells requiring refinement in each level are grouped together into rectangular blocks which are uniformly refined. This means that some coarse grid cells may be unnecessarily refined, but has the advantage that all grids are uniform and rectangular. This allows us to maintain vectorization without using gather/scatter operations. It also allows for a simple user interface, since a finite difference scheme can be written for a uniform rectangular grid without concern for the connectivity of each cell. The use of fine grids instead of unstructured grid points also reduces the storage overhead, which is on a per grid basis for our method, rather than the overhead per cell found in unstructured mesh calculations. The additional complications introduced by this approach occur at the interfaces of the fine and coarse grids (see below).

In addition to refining the spatial grid for time-accurate computations we use a smaller time step on the fine grids as well. This keeps the mesh ratio of time step to space step the same on all grids, and so the same explicit finite difference scheme is stable on all grids. The computational work is thus further concentrated on the fine grids, where it should be. In contrast, some adaptive methods for transient flows use the same time step for the whole mesh [16,17]. This can be less efficient, since the resulting Courant number may be far smaller than necessary over the unrefined portion of the grid.

AMR uses an automatic error estimation procedure, based on Richardson extrapolation, to determine the regions in the domain where the resolution in the solution is insufficient. These coarse grid cells are "flagged" as needing refinement. In addition, the irregular grid cells at solid bodies should be flagged as needing refinement if the geometry of the boundary is under-resolved. An automatic grid generation algorithm groups these flagged cells into rectangular grid patches. We have developed heuristic procedures that are quite successful at this type of grid generation [4]. We try to balance the conflicting goals of minimizing the number of fine grids and minimizing the area that is unnecessarily refined.

The time accurate integration algorithm proceeds by taking one step on the coarsest grid, and as many steps as necessary on the finer level grids until they catch up to the coarse grid time. If there are several levels of fine grids, this is applied recursively. At this point the grids are advanced independently of each other, except that fine grids require boundary values from adjacent fine grids or interpolated from the coarser grids. For a five point stencil, a fine grid will need 2 points all around the outside of the grid in order to advance the solution one step. If there is an adjacent fine grid, it can supply the missing points. Otherwise, these so-called dummy points are obtained using bilinear interpolation in space and linear interpolation in time from the coarse grid.

Since we will be computing discontinuous solutions of hyperbolic conservation laws, the adaptive mesh refinement algorithm needs to be conservative. This is complicated by the use of different time steps on the different grids. Conservation is ensured in three different parts of the mesh refinement algorithm. When two adjacent levels of grids are at same time, the fine grid updates the coarse grid, performing the conservative averaging procedure

$$U_{i,j}^{coarse} = \sum_{m=0}^{r-1} \sum_{n=0}^{r-1} U_{k+m, l+n}^{fine}$$

where r is the mesh refinement ratio, for each coarse cell

(i,j) containing a fine grid cell (k,l) in the lower left corner. If a fine grid is then removed, the total mass in the domain is conserved. Secondly, after every integration step the solution is post-processed at all coarse grid points adjacent to a fine grid. The initial coarse flux (computed ignoring the fine grids) is subtracted, and the sum of the fine grid fluxes over space and time is added in its place. Thirdly, we use conservative interpolation procedures to initialize the solution when a fine grid is created. A more complete description of the algorithm for time-dependent pdes is in [6].

5. Numerical Results

We illustrate the method on two time-dependent problems involving shock waves. In the first example we compute flow around two cylinders. An incident shock travels at Mach number 2.81. One cylinder is slightly ahead of the other. This leads to an interesting pattern of wave reflections between the two cylinders. In addition there is a reflected bow shock, and complicated wave structures behind the cylinders after the shocks pass by. All of these regions use the adaptive refinement as the solution develops. Figure 4 shows the incident shock with the location of the refined grids indicated. The initial coarse grid is 64 by 64. Two levels of grids are used, with a refinement factor of 4. Figure 5 shows density contours of the solution at later stages of the simulation.

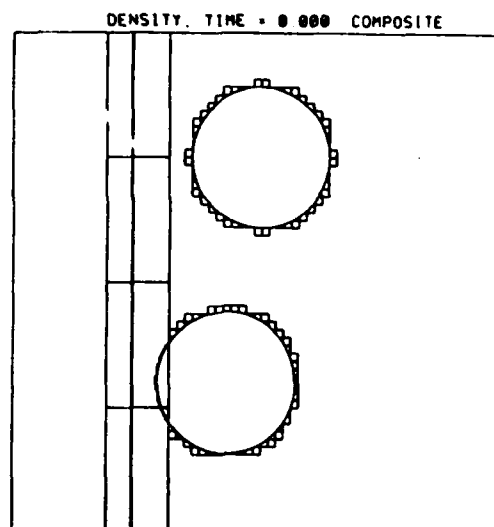


Figure 4 Density contours and grid locations at initial time for shock impinging on two cylinders.

The second example we consider is a Mach 2.2 shock travelling in a channel with a 90 degree bend. This has previously been studied in [1,21]. We use a very coarse initial grid, since much of the initial rec-

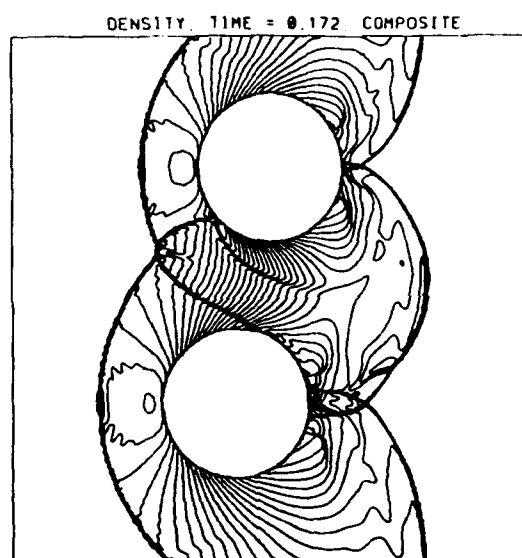
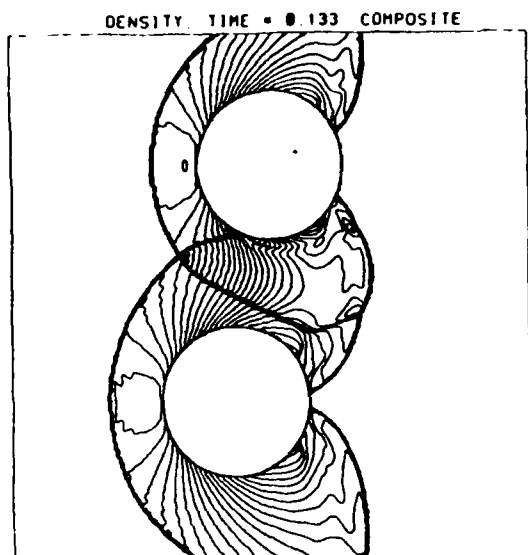


Figure 5 Density contours at later times.

tangular grid is outside the computational domain. We use two additional levels of refinement by a factor of 4 in each case in order to obtain good resolution of the shock and induced wave pattern. Figure 6 shows density contours when the shock has passed most of the way through the channel. In this figure, the location of the three levels of refined grids is indicated on the contour plots.

6. Conclusions

This work demonstrates the feasibility of an adaptive Cartesian grid approach for fluid problems in complicated geometries. Several aspects of this approach are still under development. We would like to improve the boundary scheme to make it second order along with the

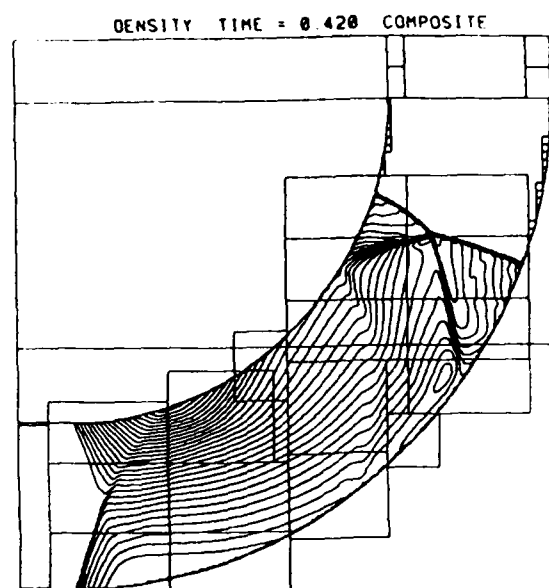


Figure 6 Density contours for shock in a channel with 90 degree bend.

interior scheme. We also hope to eliminate the smoothing step now used to insure stability in the very small cells.

We also plan to include an acceleration procedure for steady state calculations. For nested Cartesian grids multigrid is particularly attractive, since the data structures and grid transfer operations in the adaptive grid refinement algorithm make up almost all of those needed in multigrid [5]. Local grid refinement and multigrid have already been combined using a logically rectangular body-fitted grid in [7].

An important consideration is whether these techniques will extend to Navier-Stokes calculations. For problems with boundary layers, it is usually desirable to use body-fitted grids and refine heavily in the direction normal to the boundary. Refining Cartesian grid cells near such a boundary may be highly inefficient. In such cases a component grid approach may be useful, in which there are several grids with distinct coordinate systems. For example, there may be a thin body-fitted boundary layer region in addition to an underlying Cartesian grid. These multiple components will overlap in an arbitrary way, creating small irregular cells as in the Cartesian mesh method above. Again, stable and conservative difference equations are needed to compute the flow at these mesh junctions. The techniques used here should be directly applicable to this situation. Such an approach has previously been considered by others, e.g. [2,3]. However, these previous efforts did not treat the interface conditions between the different grids in an accurate or conservative way.

Acknowledgements

We thank Jeff Saltzman for the use of his high order Godunov integrator. The first author's work was supported in part by DOE Contract No. DEAC0276ER03077-V, by the AFOSR under Contract No. 86-0148. The authors were supported by NSF Presidential Young Investigator Award ASC-8858101 and DMS-8657319.

6. References

- [1] T. Aki, "A Numerical Study of Shock Propagation in Channels with 90 Degree Bends", National Aerospace Laboratory Technical Report, Tokyo, Japan, 1987.
- [2] E. Aua, "Component-Adaptive Grid Interfacing", AIAA Paper No. 81-0382. Proc. AIAA 19th Aerospace Sciences Meeting, 1981.
- [3] J. Benek, J. Steger and F. Dougherty, "A Flexible Grid Embedding Technique with Application to the Euler Equations", AIAA Paper No. 83-1944. Proc. 6th Computational Fluid Dynamics Conference, Danvers, Mass. July, 1983.
- [4] M.J. Berger, "Data Structures for Adaptive Grid Generation", SIAM J. Sci. Stat. Comp. 7, (1986), pp. 904-916.
- [5] M.J. Berger, "Adaptive Finite Difference Methods in Fluid Dynamics". Lecture series 1987-04 in Computational Fluid Dynamics at the von Karman Institute for Fluid Dynamics, Rhode Saint Genese, Belgium, March, 1987.
- [6] M. Berger and P. Colella, "Local Adaptive Mesh Refinement for Shock Hydrodynamics". To appear in J. Comp. Phys.
- [7] M.J. Berger and A. Jameson, "Automatic Adaptive Grid Refinement for the Euler Equations", AIAA J. 23, (1985), pp. 561-568.
- [8] M. Berger and J. Oliger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations", J. Comp. Phys. 53 (1984), pp. 484-512.
- [9] I. Chern and P. Colella. "A Conservative Front Tracking Method for Hyperbolic Conservation Laws". Submitted to J. Comp. Phys. UCRL-97200, July 1987.
- [10] S. Choi and B. Grossman, "A Flux-Vector Split, Finite-Volume Method for Euler's Equations on Non-Mapped Grids", AIAA Paper 88-0227, Proc. 26th Aerospace Sciences Meeting, Reno, Nevada.
- [11] D. Clarke, H. Hassan, and M. Salas, "Euler Calculations for Multielement Airfoils Using Cartesian Grids, AIAA Paper 85-0291, Jan. 1985.
- [12] R. Gaffney, H. Hassan and M. Salas, "Euler Calculations for Wings Using Cartesian Grids", AIAA Paper 87-0356, January 1987.
- [13] R.J. LeVeque, "A Large Time Step Generalization of Godunov's Method for Conservation Laws", SIAM J. Num. Anal. 22 (1985), pp. 1051-1073.
- [14] R.J. LeVeque, "High Resolution Finite Volume Methods on Arbitrary Grids via Wave Propagation", J. Comp. Phys. 78 (1988), pp. 36-63.
- [15] R.J. LeVeque, "Cartesian Grid Methods for Flow in Irregular Regions", in Numerical Methods for Fluid Dynamics III, K.W. Morton and M.J. Baines, editors, Clarendon Press (1988), pp. 375-382.
- [16] Lohner, "Finite Elements in CFD: What Lies Ahead". Proc. World Congress on Computational Mechanics, Austin, Texas. Sept. 1986.
- [17] J. Saltzman and J. Brackbill, "Applications and Generalizations of Variational Methods for Generating Adaptive Meshes", in Numerical Grid Generation, J. Thompson, ed., North-Holland, 1982.
- [18] Usab and Murman, "Embedded Mesh Solutions of the Euler Equations Using a Multiple-grid Method", AIAA Paper 83-1946-CP, 6th AIAA Computational Fluid Dynamics Conference, Danvers, Mass. July 1983.
- [19] B. Wedan and J. South, "A Method for Solving the Transonic Full-Potential Equations for General Configurations", Proc. AIAA Computational Fluid Dynamics Conf., July 1983.
- [20] P. Woodward, Proc. Nato Workshop in Astrophysical Radiation Hydrodynamics, Munich, W. Germany, Nov. 1983. Also Lawrence Livermore Report UCRL-90009, August, 1982, (unpublished).
- [21] H. Yee, "Upwind and Symmetric Shock-Capturing Schemes", NASA Technical Memorandum 89464, May 1987.